



Escuela
Politécnica
Superior

Segmentación de objetos urbanos 3D usando Lidar



Grado en Ingeniería Robótica

Trabajo Fin de Grado

Autor:

Felipe Ruiz Berná

Tutor/es:

Miguel Ángel Cazorla Quevedo

Francisco Gómez Donoso

Enero 2021



Universitat d'Alacant
Universidad de Alicante

Segmentación de objetos urbanos 3D usando Lidar

Autor

Felipe Ruiz Berná

Tutor/es

Miguel Ángel Cazorla Quevedo

CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

Francisco Gómez Donoso

CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL



Grado en Ingeniería Robótica



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Enero 2021

Preámbulo

El hecho de elegir el Deep Learning a la hora de desarrollar mi Trabajo de Fin de Grado está motivado, en gran parte, a las innumerables aplicaciones que este campo posee. Esto despertó en mí un gran interés, el cual aumentaba exponencialmente con cada artículo nuevo que leía sobre este tema.

Mi Trabajo de Fin de Grado consiste en aplicar Deep Learning en conducción autónoma. Concretamente, este proyecto tiene como finalidad la detección de objetos 3D a partir de los datos obtenidos por un LiDAR.

En primer lugar, me gustaría agradecer a mis tutores la ayuda recibida y el tiempo que han invertido en este trabajo, especialmente por la dura situación en la que se enmarca el desarrollo de este proyecto debido a la pandemia de COVID-19. Desde el primer contacto que tuvimos, han demostrado una gran predisposición por ayudarme a realizar un proyecto innovador y de calidad, manteniendo una fluida comunicación semana tras semana y proporcionándome las herramientas necesarias para ello. Es por ello que les dedico estos agradecimientos.

Por supuesto debo agradecer a toda mi familia el apoyo y esfuerzo depositado en mí, acompañándome con tanto cariño durante esta etapa universitaria y, en general, toda la vida. Sin ellos no sería quién soy.

Además, merece unas palabras especiales mi madre, pilar fundamental en mi vida y a quien debo los valores fundamentales que rigen mi personalidad. Gracias por dar tu vida por mí, por priorizar mis necesidades a las tuyas y por tu amor incondicional.

A mis amigos, por los buenos momentos que hemos compartido juntos desde hace tantos años y, en especial, a quien comenzó, fue mi acompañante, y terminó esta carrera junto a mí. Solamente tú y yo sabemos lo que hemos pasado estos años y, sin ti, no hubiera sido lo mismo.

Por último, quiero dedicar unas palabras de agradecimiento a una persona especial que me ha acompañado prácticamente desde el principio de este viaje, permaneciendo conmigo en las largas sesiones de estudio año tras año y, finalmente, en la redacción de esta tesis. El cariño y amor que me demuestras día a día es infinito y nunca podré agradecerte lo suficiente todo lo que haces por mí. Gracias, contigo todo es más sencillo.

*No te conformes con el mundo que has heredado.
Nunca se ha resuelto un desafío
sin personas que pensasen diferente*

Tim Cook.

Índice general

1	Introducción	1
2	Marco Teórico	5
2.1	Hardware	5
2.1.1	LiDAR	6
2.2	Software	6
2.2.1	Actuales modelos de detección de objetos 3D	8
3	Objetivos	13
4	Metodología	15
4.1	Setup	15
4.2	Datasets	15
4.2.1	KITTI Dataset	15
4.2.1.1	Setup de los sensores	16
4.2.1.2	Anotaciones	17
4.2.2	Dataset USYD CAMPUS DATASET	17
4.3	Arquitectura del modelo	18
4.3.1	PV-RCNN	18
4.3.2	Nuestro modelo basado en PV-RCNN	19
4.4	ROS	22
4.4.1	Sistema de archivos	22
4.4.2	Grafo de computación	23
5	Desarrollo	25
5.1	Implementación de la red en PyTorch y prueba con el dataset KITTI	25
5.2	Procesamiento del USYD CAMPUS DATASET y prueba del modelo	27
5.2.1	Bag2Pcd	28
5.2.2	Pcd2Bin	28
6	Resultados	31
6.1	Explicación de los datos y los parámetros usados	31
6.2	Visualización de los resultados	33
6.2.1	Resultados en el dataset KITTI	34
6.2.2	Resultados en USYD CAMPUS DATASET	36
6.2.3	Problemas del modelo con el USYD CAMPUS DATASET	36
6.2.4	Soluciones al problema presentado	36
7	Conclusiones	39

Bibliografía**41**

Índice de figuras

1.1	Grados de autonomía en la conducción	2
1.2	Arquitectura de AlexNet	3
2.1	HW y SW de la plataforma de conducción autónoma	5
2.2	Visualización de los resultados de YOLO	7
2.3	Funcionamiento de F-PointNet	9
2.4	Resultados de Vote3Deep	10
4.1	Setup de los sensores. Se observa la disposición de los sensores montados en el automóvil para obtener los datos.	16
4.2	Coordenadas del objeto. Se ilustra el sistema de coordenadas de los bounding boxes 3D anotados con respecto al sistema de coordenadas del escáner láser Velodyne 3D. En la dirección Z, el sistema de coordenadas del objeto está situado en la parte inferior del objeto (punto de contacto con la superficie de apoyo)	17
4.3	Arquitectura de PV-RCNN	19
4.4	Arquitectura de nuestra aproximación	20
4.5	Arquitectura del módulo de deformación adaptable.	21
4.6	Arquitectura del módulo de fusión de contextos.	21
4.7	Sistema de archivos de ROS	22
4.8	Grafo computacional de ROS	24
5.1	Diagrama general del desarrollo	29
6.1	Impacto del <i>Learning Rate</i> en el aprendizaje	33
6.2	Primeros resultados del modelo en USYD CAMPUS DATASET	37
6.3	Resultados del modelo en USYD CAMPUS DATASET tras reentrenar el modelo. Ejemplo 1.	38
6.4	Resultados del modelo en USYD CAMPUS DATASET tras reentrenar el modelo. Ejemplo 2.	38

Índice de tablas

2.1	Sensor LiDAR y visualización de los datos obtenidos	6
2.2	Resultados del modelo CLOCs PVCas	8
2.3	Resultados de detección de coches de CLOCs PVCas	9
2.4	Resultados de detección de peatones de F-PointNet	9
2.5	Resultados de detección de objetos de Vote3Deep	10
2.6	Resultados de detección de ciclistas de MMLab PV-RCNN	11
4.1	Visualización de algunas nubes del USYD CAMPUS DATASET con Open3D.	18
5.1	Visualización del USYD CAMPUS DATASET en RViz.	27
6.1	Distribución de las etiquetas de los objetos en el Dataset KITTI.	31
6.2	Distribución de las apariciones de los objetos <i>Coche</i> y <i>Camión</i> en el Dataset KITTI.	32
6.3	Resultados de la detección en escenas con coches y peatones	34
6.4	Resultados de la detección en escenas con ciclistas	35
6.5	Resultados de detección de objetos del modelo de las principales clases del dataset KITTI	36

Índice de Códigos

5.1	Implementación del modelo	26
5.2	Construcción del optimizador	26
5.3	Entrenamiento de la red	26

1 Introducción

La conducción autónoma es la capacidad que tienen los vehículos para poder guiarse por sí mismos y sin intervención humana hasta un destino preestablecido. Además, todos los vehículos están clasificados según el nivel de autonomía que poseen en función de la asistencia a la conducción que ofrecen al conductor. Estos niveles de autonomía son seis y, junto con las definiciones clave de estas tecnologías, han sido desarrollados por la Sociedad de Ingenieros de la Automoción (SAE).

- **Nivel 0: Sin automatización de la conducción.** La conducción del vehículo se realiza íntegramente por el conductor, aunque se incluyen en este nivel los vehículos que poseen asistentes a la conducción que no realizan el control longitudinal o lateral del vehículo, como por ejemplo la detección de vehículos en el ángulo muerto de los retrovisores.
- **Nivel 1: Asistencia al conductor.** En este nivel se incluyen asistentes a la conducción con funciones que, al contrario que en el anterior nivel, regulan el control longitudinal y lateral del vehículo aunque nunca de manera simultánea. Algún ejemplo de este nivel sería el programador de velocidad activo (ACC), que es capaz de mantener una velocidad constante y reducirla en caso de ser necesario o el asistente de ayuda al aparcamiento, que controla la dirección mientras que el conductor sigue controlando los pedales.
- **Nivel 2: Automatización parcial de la conducción.** En este nivel se añade la simultaneidad en el control lateral y longitudinal del vehículo. Sin embargo, el conductor sigue siendo el responsable de la conducción, ya que estos sistemas tienen un uso limitado y los vehículos no están preparados para reaccionar ante un obstáculo imprevisto. Un ejemplo de este nivel es el sistema de mantenimiento en el centro del carril junto con un ACC.
- **Nivel 3: Automatización condicional de la conducción.** En este nivel se produce un gran salto cualitativo, ya que el conductor puede decidir que el sistema de conducción autónoma realice la totalidad de las funciones de la conducción, aunque con unas ciertas limitaciones. Así pues, el conductor ya no necesita supervisar la conducción, pero debe estar presente y alerta para intervenir ante el aviso del sistema, lo cual puede suceder por varios motivos, como por ejemplo una situación de riesgo ante la que el sistema no sabe reaccionar.
- **Nivel 4: Alta automatización de la conducción.** En este nivel se elimina la necesidad de que el conductor responda ante una demanda del sistema, ya que se prevé que el sistema pueda mantener la conducción de una manera sostenida en el tiempo para guiar al vehículo hasta su destino. Tan solo en ciertas ocasiones muy específicas cuando se encuentra fuera de su ámbito de funcionamiento sería necesario la intervención del

conductor, como por ejemplo al sobrepasar zonas delimitadas geográficamente en el software del vehículo.

- **Nivel 5: Automatización total de la conducción.** En este último nivel, el sistema de conducción autónoma está diseñado para tener un ámbito de funcionamiento que comprende cualquier circunstancia, condiciones y lugares por los que podría circular un humano, ya que no existen limitaciones climáticas o geográficas, por lo que se puede prescindir del volante y los pedales al no necesitar conductor humano.

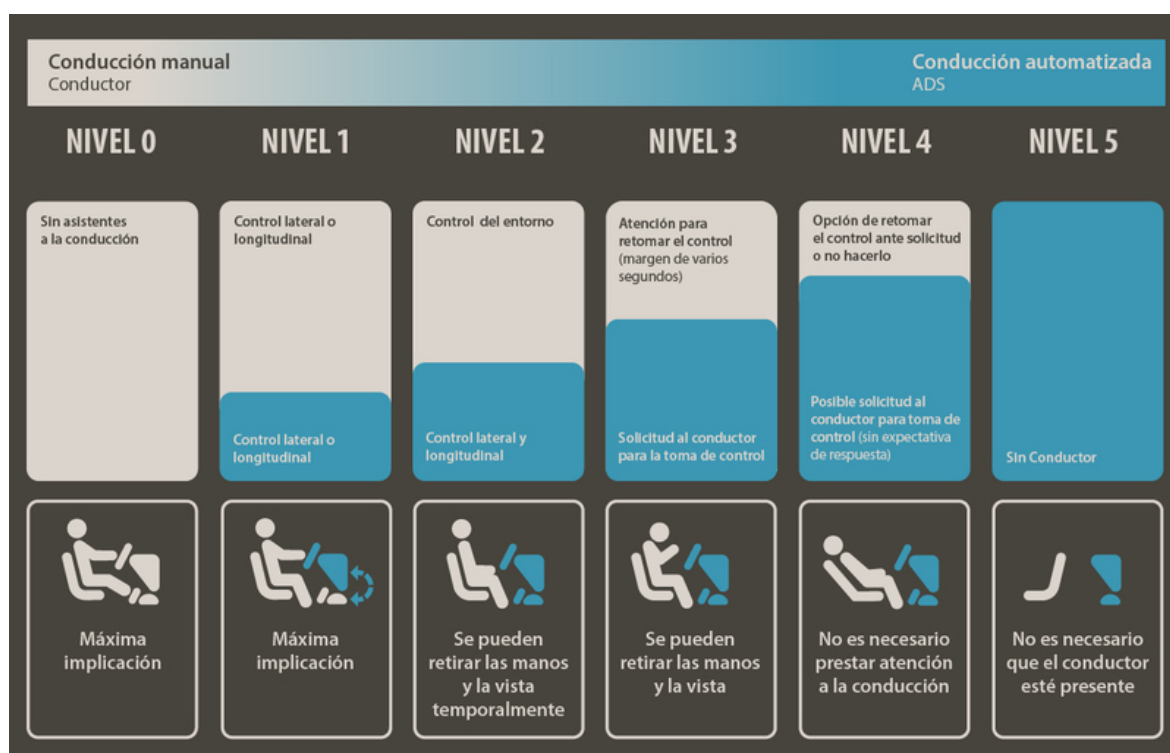


Figura 1.1: Grados de autonomía en la conducción

Para desarrollar un sistema de conducción autónoma se necesita nutrir a los vehículos con distintos sensores, puesto que estos serán los "sentidos" del vehículo. Actualmente se usan distintos tipos de sensores en función de la rama de investigación por la que se apueste y el nivel de equipamiento, entre los que destacan los sensores ultrasónicos, sistemas de navegación inercial, sensores infrarrojos, sistema de cámaras, Radar y, por último, Lidar.

La mayoría de las grandes marcas de fabricación de automóviles como pueden ser Ford, Toyota, Nissan, Volvo o Volkswagen apuestan actualmente por la inclusión del Lidar en los vehículos autónomos, mientras que en el extremo opuesto se encuentra Tesla, el cual apuesta por el uso de cámaras junto con otros sensores específicos para ciertas tareas de la conducción para reducir el coste de producción de los vehículos.

Dejando a un lado el hardware utilizado y adentrándonos en el mundo del software, tanto los fabricantes que apuestan por Lidar como los que apuestan por sistemas de cámaras 2D,

centran sus esfuerzos en el desarrollo de técnicas de Deep Learning para mejorar la fiabilidad de los vehículos a través de la detección de objetos. Aunque se lleva investigando en Deep Learning desde mediados del siglo pasado, se ha producido un boom desde el año 2012, el cual ha sido provocado principalmente por la introducción de nuevas redes neuronales con estructuras específicas (Redes Convolucionales) y el aumento de la capacidad de procesamiento de los computadores, incluyendo en ellos unidades especializadas (GPUs).

Las técnicas basadas en Deep Learning usadas para detección de objetos en imágenes están muy avanzadas, hasta el punto de que se considera resuelto el problema de clasificación de objetos en imágenes en ImageNet Large Scale Visual Recognition Challenge (ILSVRC), superado con creces en 2012 por el modelo AlexNet de Alex Krizhevsky. Sin embargo, el problema de la conducción autónoma no está resuelto ni está cerca de resolverse debido en parte a temas como los fallos de seguridad, la ética o el alto precio de los sensores.

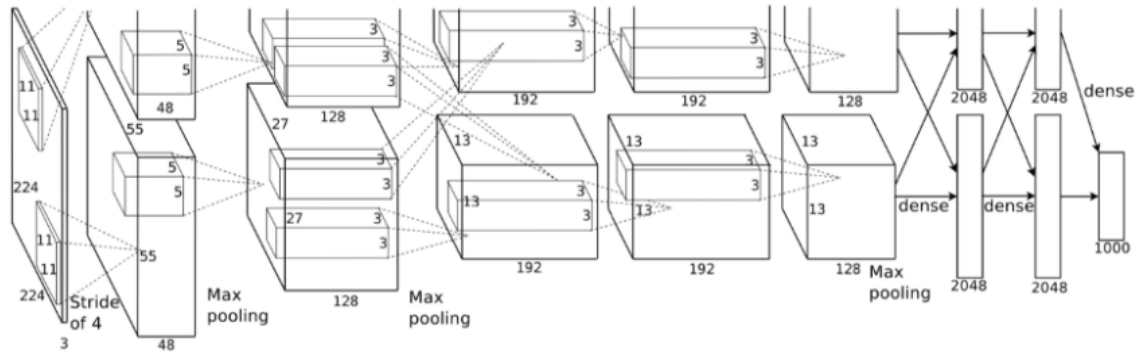


Figura 1.2: Arquitectura de AlexNet

Tras lo expuesto anteriormente, este proyecto se centra, por tanto, en el desarrollo de ciertas técnicas y tecnologías basadas en Deep Learning y los datos 3D obtenidos de los Lidar para intentar alcanzar el último nivel de autonomía en la conducción a través de la detección de objetos habituales en el entorno de la conducción, como pueden ser coches, camiones, peatones, ciclistas...

2 Marco Teórico

En esta sección voy a exponer el marco teórico sobre el problema de detección de objetos 3D en conducción autónoma. Aunque existen modelos de detección de objetos 3D en ambientes *indoor*, los cuales se centran en el reconocimiento de objetos tales como mesas, sillas, sofás o puertas, voy a centrarme concretamente en ambientes *outdoor*, ya que los datasets utilizados en este proyecto están obtenidos en exteriores.

Para entender cómo la detección de objetos 3D va a ayudar a mejorar la experiencia de conducción autónoma, primero necesitamos entender los aspectos clave en este tema.

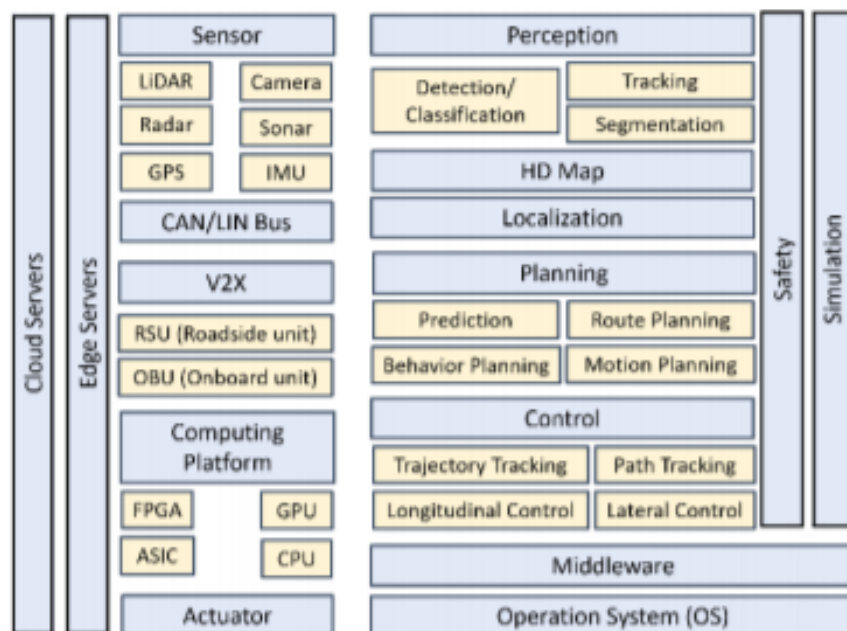


Figura 2.1: HW y SW de la plataforma de conducción autónoma

2.1 Hardware

Como hemos comentado anteriormente, gran parte del desarrollo en conducción autónoma se basa en la elección de los sensores utilizados para tal fin. Podemos definir los sensores que se encuentran en una plataforma móvil en 3 categorías principales: Sensores propioceptivos, sensores de localización y sensores exteroceptivos.

- **Sensores propioceptivos:** Son sensores que captan la información propia del vehículo, como por ejemplo la velocidad, aceleración, ángulo de giro, etc. También se incluyen en

esta categoría IMUs y giroscopios.

- **Sensores de localización:** Utilizan sensores externos como GPS para determinar la posición y orientación global del vehículo.
- **Sensores extereoceptivos:** Son capaces de percibir las líneas de la carretera, señales de tráfico, obstáculos o condiciones ambientales.

Además, esta serie de sensores pueden dividirse en sensores activos y pasivos. Los pasivos basan su funcionamiento en percibir las ondas electromagnéticas presentes en el entorno, mientras que los sensores activos emiten las ondas electromagnéticas para medir el tiempo que tarda en volver esa onda para calcular distintos parámetros. En esta clasificación de sensores activos se encuentra el LiDAR, sensor en el cual se centra nuestro proyecto.

2.1.1 LiDAR

Un LiDAR (Light Detection and Ranging) es un dispositivo capaz de determinar la distancia a un objeto o superficie utilizando un haz láser pulsado. Esta distancia se calcula midiendo el tiempo que tarda el pulso en viajar hasta el objeto y volver al dispositivo. Se usa principalmente en geología y sismología, pero en los últimos años se está empleando para la investigación en conducción autónoma de vehículos.

Este sistema es ideal para la detección de objetos, ya que nos permite obtener una nube de puntos 3D de la escena mediante un escáner láser. Para realizar este escaneo se combinan un movimiento longitudinal dado por la trayectoria del vehículo y otro transversal mediante un espejo móvil que desvía el haz de luz emitido por el escáner. Así pues, obtenemos miles de puntos por segundo, conformando nubes 3D que nos servirán de entrada para nuestro proyecto.

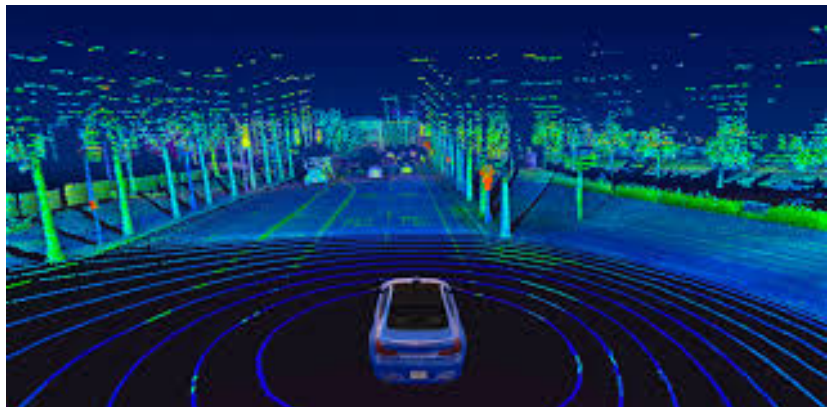


Tabla 2.1: Sensor LiDAR y visualización de los datos obtenidos

2.2 Software

En un vehículo autónomo el software empleado debe trabajar en tiempo real, y su arquitectura puede ser *end-to-end* o de estilo modular.

- Los sistemas con una arquitectura *end-to-end* generan las señales de control directamente desde la entrada de datos de los sensores. Estas señales de control pueden ser el manejo de la dirección y los pedales (aceleración y freno). Además, hay dos aproximaciones principales para este tipo de sistemas: Deep Learning supervisado y Aprendizaje por refuerzo.
- Los sistemas modulares en cambio están contruidos como un *pipeline* de múltiples componentes que conectan los datos obtenidos por los sensores con los actuadores finales. Las principales funciones que se encuentran en este *pipeline* son percepción, localización y mapeo, predicción, generación de rutas y control del vehículo.

Además, existen distintas tecnologías para realizar las tareas de percepción del vehículo, dependiendo del hardware utilizado. En caso de apostar por las cámaras 2D, se deberá desarrollar un sistema de procesamiento de imágenes y detección 2D, donde actualmente destaca claramente el modelo *YOLO* y sus posteriores actualizaciones.

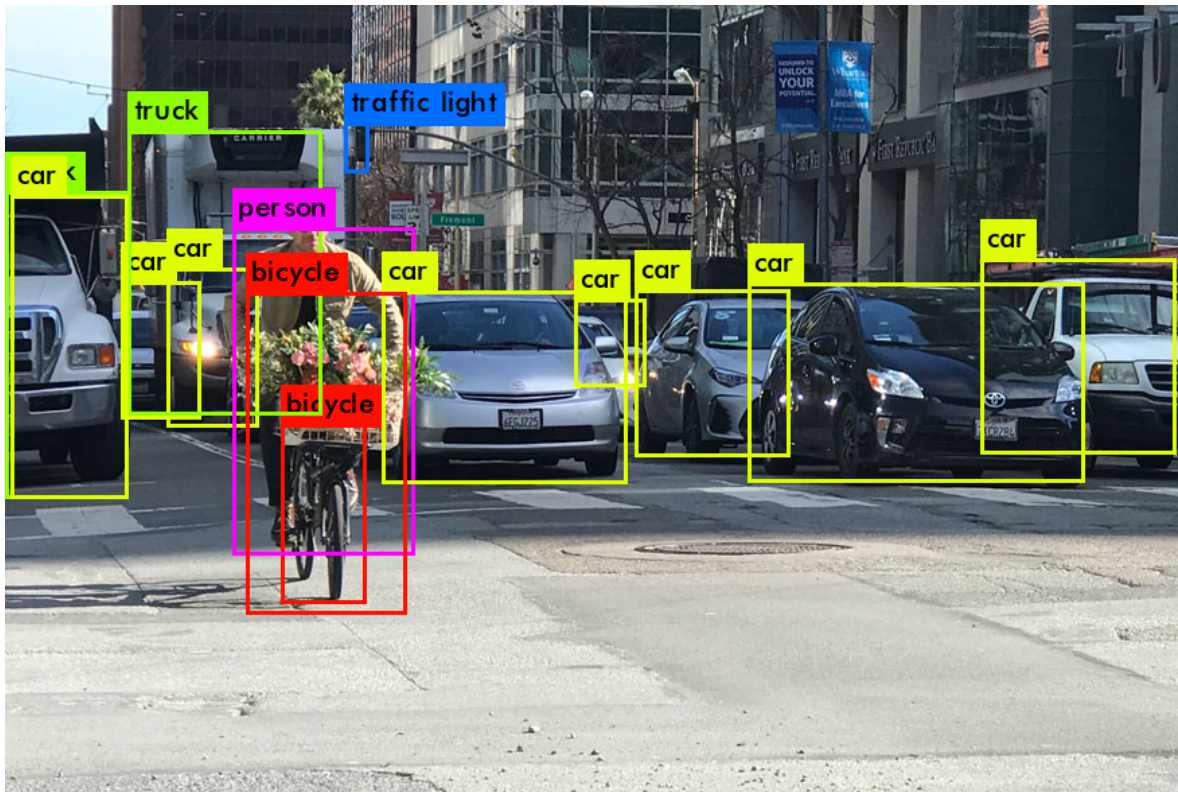


Figura 2.2: Visualización de los resultados de YOLO

En caso de apostar por sensores 3D como el LiDAR, el modelo empleado para la detección se nutrirá con las escenas en 3D generadas por el sensor y, en la mayoría de casos, producirá como resultado unos *bounding boxes* que contendrán al objeto detectado en cuestión. Puesto que este proyecto se centra en la detección 3D, veamos más a fondo los modelos que actualmente se encuentran a la vanguardia en esta tarea.

2.2.1 Actuales modelos de detección de objetos 3D

En los ambientes exteriores, por donde circularían los vehículos autónomos, se suele tomar como referencia el dataset KITTI, el cual es un dataset de nubes de puntos de objetos urbanos con sus correspondientes imágenes RGB, cuyas principales clases para detección son coches, peatones y ciclistas. En base a este dataset se han desarrollado multitud de modelos aplicando diversas técnicas y tecnologías, siendo los más destacados (en función de los resultados obtenidos en dificultad moderada para cada una de las principales clases) los siguientes:

- **Detección de coches:**

- **CLOCs PVCas:** Este modelo supera la dificultad que había hasta el momento para entrenar redes con datos 3D y 2D de manera simultánea, proponiendo una novedosa red de fusión de candidatos a objetos Cámara-LiDAR (CLOCs por sus siglas en inglés). La fusión de estos CLOCs proporciona un framework de fusión multimodal de baja complejidad que mejora significativamente el rendimiento de los detectores de una única modalidad.

CLOCs funciona con los candidatos de salida antes de aplicar Non-Maximum Suppression (NMS) y está entrenado para aprovechar las consistencias geométricas y semánticas y obtener así resultados más precisos en detección 3D y 2D. Además, muestra un gran avance en detección de objetos a larga distancia en comparación con otros modelos estado del arte que trabajan con este dataset.



Tabla 2.2: Resultados del modelo CLOCs PVCas

Como podemos observar en la siguiente tabla, este modelo obtiene un nivel de precisión altísimo, lo que lo coloca a la cabeza en modelos de detección 3D de coches en el challenge de KITTI.

Benchmark	Easy	Moderate	Hard
Car (Detection)	96.76 %	95.96%	91.08 %
Car (Orientation)	96.74 %	95.79 %	90.81 %
Car (3D Detection)	88.94 %	80.67 %	77.15 %
Car (Bird's Eye View)	93.05 %	89.80 %	86.57 %

Tabla 2.3: Resultados de detección de coches de CLOCs PVCas

- **Detección de peatones:**

- **F-PointNet:** Este modelo basa gran parte de su funcionamiento y precisión en la detección 2D (con todos los problemas que esto puede conllevar: oclusiones, mala luminosidad...), ya que obtiene los datos con una cámara RGB-D y trabaja creando los bounding boxes 2D candidatos utilizando una red neuronal convolucional 2D. Tras esto, se obtiene una nube de puntos con el objeto. Después, esta nube sirve de entrada a una red PointNet, la cual predice la clase del objeto.

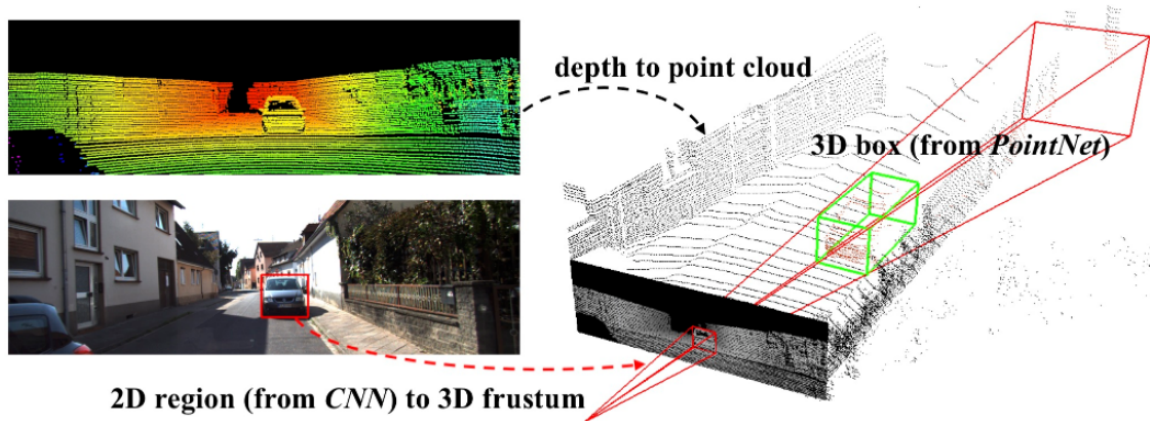


Figura 2.3: Funcionamiento de F-PointNet

En un entorno de dificultad moderada, los resultados en detección de peatones de este modelo son muy buenos, habiendo llegado a ocupar el primer puesto en el challenge de KITTI en el apartado de detección de peatones.

Benchmark	Easy	Moderate	Hard
Pedestrian (Detection)	89.83 %	80.13 %	75.05 %
Pedestrian (3D Detection)	50.53 %	42.15 %	38.08 %
Pedestrian (Bird's Eye View)	57.13 %	49.57 %	45.48 %

Tabla 2.4: Resultados de detección de peatones de F-PointNet

- **Vote3Deep:** Este es otro de los modelos destacados en detección de peatones, el cual trabaja con nubes de puntos representadas como una malla 3D, utilizando una red neuronal convolucional y un algoritmo de voto que consiste en realizar las convoluciones únicamente a los elementos que no son cero, por lo que se obtiene una mayor eficiencia que aplicando una red convolucional clásica. Aunque este modelo, con una pequeña variación del kernel de la última capa, también se usa para la detección de coches y ciclistas, los resultados donde destaca es en la detección de peatones. Sin embargo, utilizar el mismo modelo para la detección de las tres clases afecta a su precisión, donde se observa que la precisión en la detección de peatones desciende considerablemente.

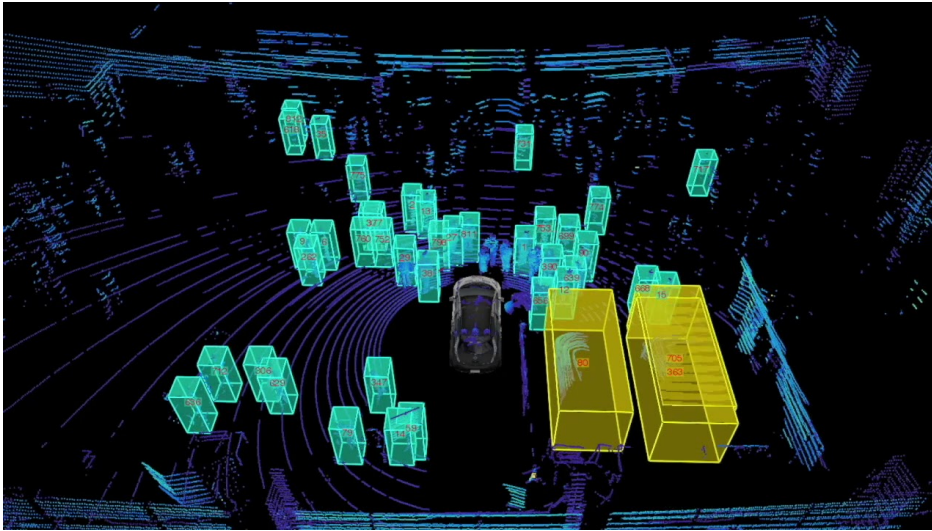


Figura 2.4: Resultados de Vote3Deep

Benchmark	Easy	Moderate	Hard
Car (Detection)	78.95 %	70.30 %	63.12 %
Pedestrian (Detection)	67.99 %	54.80 %	51.17 %
Cyclist (Detection)	78.41 %	68.82 %	62.50 %

Tabla 2.5: Resultados de detección de objetos de Vote3Deep

- **Detección de ciclistas:**

- **MMLab PV-RCNN:** Esta aproximación es muy interesante, ya que utiliza únicamente datos 3D obtenidos por LiDAR, y consigue una gran precisión basándose en el framework de detección de objetos 3D PV-RCNN y en un conjunto de modelos simples con NMS y el algoritmo de voto explicado anteriormente.

Para obtener estos buenos resultados, se han producido modificaciones en PV-RCNN como la voxelización dinámica, información temporal o selección de mues-

tras de entrenamiento adaptativo.

Benchmark	Easy	Moderate	Hard
Cyclist (Detection)	86.62 %	80.42 %	73.64 %
Cyclist (Orientation)	86.43 %	79.70 %	72.96 %
Cyclist (3D Detection)	78.60 %	63.71 %	57.65 %
Cyclist (Bird's Eye View)	82.49 %	68.89 %	62.41 %

Tabla 2.6: Resultados de detección de ciclistas de MMLab PV-RCNN

3 Objetivos

Con este Trabajo Final de Grado (TFG) se alcanzan los siguientes objetivos:

- **Entender la metodología del Deep Learning:** Nociones teóricas sobre las redes neuronales en general, implementación de dichas redes, conocimiento sobre distintas herramientas para desarrollo de modelos de Deep Learning (Keras, TensorFlow, PyTorch...).
- **Conocer las principales herramientas y modelos en detección de objetos 3D:** Nutrirse con las técnicas actuales utilizadas por los modelos más avanzados en reconocimiento de objetos en 3D, así como la introducción al mundo de la investigación.
- **Entender el hardware utilizado para obtener los datos y los fundamentos de calibración de sensores:** Obtención de las nubes de puntos a través de Lidar y su sincronización con las imágenes 2D obtenidas con las cámaras RGB.
- **Entender y usar los conceptos de ROS para preprocesamiento de datos:** Manejar topics, suscriptores, publicadores... para realizar el procesamiento de los datos del dataset de Sydney en formato Rosbag y convertirlos a formato KITTI.
- **Introducción a la detección 3D en conducción autónoma:** Comparación de distintas tecnologías usadas en modelos del estado del arte en tareas de conducción autónoma.

4 Metodología

En esta sección voy a explicar las herramientas, técnicas y tecnologías desarrolladas en este proyecto.

4.1 Setup

- Sistema operativo Linux (Ubuntu 20 y virtualización de Ubuntu 18 para ciertas tareas)
- ROS Noetic y ROS Melodic
- Lenguaje de programación Python 3.6
- PyTorch 1.5
- CUDA 9.0
- Spconv v1.0
- Tarjeta gráfica Nvidia GeForce GTX 1650 Max-Q

4.2 Datasets

En este apartado voy a explicar el contenido de los datasets utilizados en este proyecto. En primer lugar se utiliza el dataset de KITTI para entrenar y validar la red y, una vez obtenemos el modelo con buenos resultados, se le pasa el contenido del USYD CAMPUS DATASET para intentar realizar el etiquetado automático de este dataset.

4.2.1 KITTI Dataset

Como se ha comentado anteriormente, el dataset KITTI es el referente para detección 3D de objetos en tareas de conducción autónoma, por ello es el elegido para entrenar la red en este proyecto. El contenido básico de este dataset son nubes de puntos con sus respectivas imágenes RGB y el correspondiente etiquetado de las clases para poder llevar a cabo el entrenamiento, dividido además en archivos para entrenamiento y archivos para validación.

Este dataset fue grabado a través de un automóvil para promover la investigación en robótica móvil y conducción autónoma. En total, contiene unas 6 horas de escenarios de tráfico grabadas a 10-100Hz usando una gran variedad de sensores, como cámaras estéreo de alta resolución en color y escala de grises, un escáner láser 3D Velodyne y un sistema de navegación inercial GPS/IMU de alta precisión. Los escenarios son muy diversos y con situaciones de tráfico del mundo real, y contienen desde autopistas sobre zonas rurales hasta escenas en el centro de la ciudad, con numerosos objetos estáticos y dinámicos. Los datos de este dataset

están calibrados, sincronizados y cronometrados y, como he comentado anteriormente, se proporcionan las etiquetas de los objetos a detectar.

4.2.1.1 Setup de los sensores

Los sensores utilizados en la obtención de este dataset son los siguientes:

- 2 × PointGray Flea2 grayscale cameras (FL2-14S3M-C), 1.4 Megapixels, 1/2" Sony ICX267 CCD
- 2 × PointGray Flea2 color cameras (FL2-14S3C-C), 1.4 Megapixels, 1/2" Sony ICX267 CCD
- 4 × Edmund Optics lenses, 4mm, vertical opening angle of region of interest (ROI) 35°
- 1 × OXTS RT3003 inertial and GPS navigation system, 6 axis, 100 Hz, L1/L2 RTK, resolución: 0.02m / 0.1°
- 1 × Velodyne HDL-64E rotating 3D laser scanner, 10 Hz, 64 beams, 0.09° resolución angular, 1.3M puntos/segundo, campo de visión: 360° horizontal, 26.8° vertical, rango: 120 m

Este último sensor es el que muestra mayor interés para nuestro proyecto, puesto que trabajamos con las nubes de puntos obtenidas con este sensor.

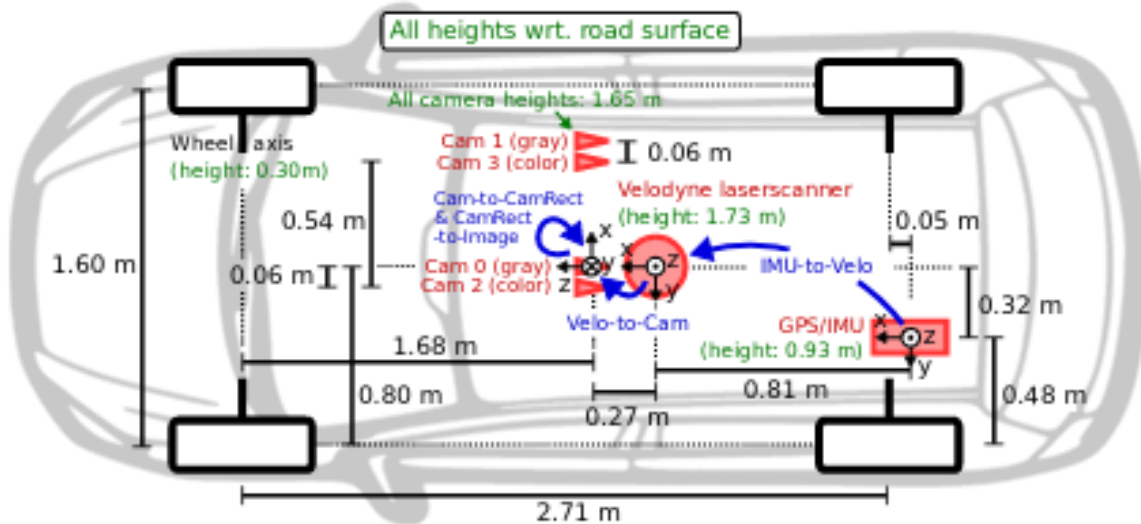


Figura 4.1: Setup de los sensores. Se observa la disposición de los sensores montados en el automóvil para obtener los datos.

4.2.1.2 Anotaciones

El dataset provee anotaciones para cada uno de los objetos dinámicos que se encuentran dentro del campo de visión de la cámara de referencia. Se definen las clases "Car", "Van", "Truck", "Pedestrian", "Person", "Cyclist", "Tram" y "Misc".

A cada objeto se le asigna una clase y su tamaño 3D (altura, anchura y longitud). Para cada frame, se proporciona la traslación y la rotación del ángulo en el eje Z del objeto en 3D, ya que la rotación en los otros ejes del vehículo se asumen a cero. Además, se especifica el nivel de oclusión de los objetos.

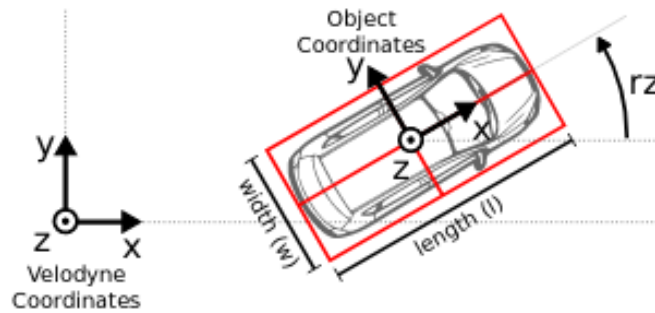


Figura 4.2: Coordenadas del objeto. Se ilustra el sistema de coordenadas de los bounding boxes 3D anotados con respecto al sistema de coordenadas del escáner láser Velodyne 3D. En la dirección Z, el sistema de coordenadas del objeto está situado en la parte inferior del objeto (punto de contacto con la superficie de apoyo)

4.2.2 Dataset USYD CAMPUS DATASET

Este dataset ha sido obtenido por un coche autónomo por la Universidad de Sidney y sus alrededores. Contiene 52 secuencias captadas por el coche a lo largo de un año para favorecer la heterogeneidad de las condiciones ambientales. Al igual que KITTI, el dataset USYD CAMPUS DATASET contiene datos de distintos sensores, tales como cámaras RGB, LiDAR Velodyne y GPS/IMU, aunque cabe destacar que el sensor LiDAR utilizado en este caso obtiene nubes de puntos menos densas, con hasta cuatro veces menos puntos que el utilizado en el de KITTI.

Otra de las diferencias con el dataset de KITTI es la falta de anotaciones y etiquetados de los objetos, por lo que no se puede usar en crudo para realizar el entrenamiento de tipo supervisado, que es el que se desarrolla en este proyecto, sino que habría que realizar un etiquetado manual de los objetos. Sin embargo, este dataset nos sirve para testear la eficacia de nuestro modelo una vez entrenado con el dataset de KITTI, comprobando además la robustez del modelo puesto que los datos de entrada tienen diferencias con respecto a los datos de KITTI, como por ejemplo la diferencia de densidad de las nubes que se ha comentado anteriormente.

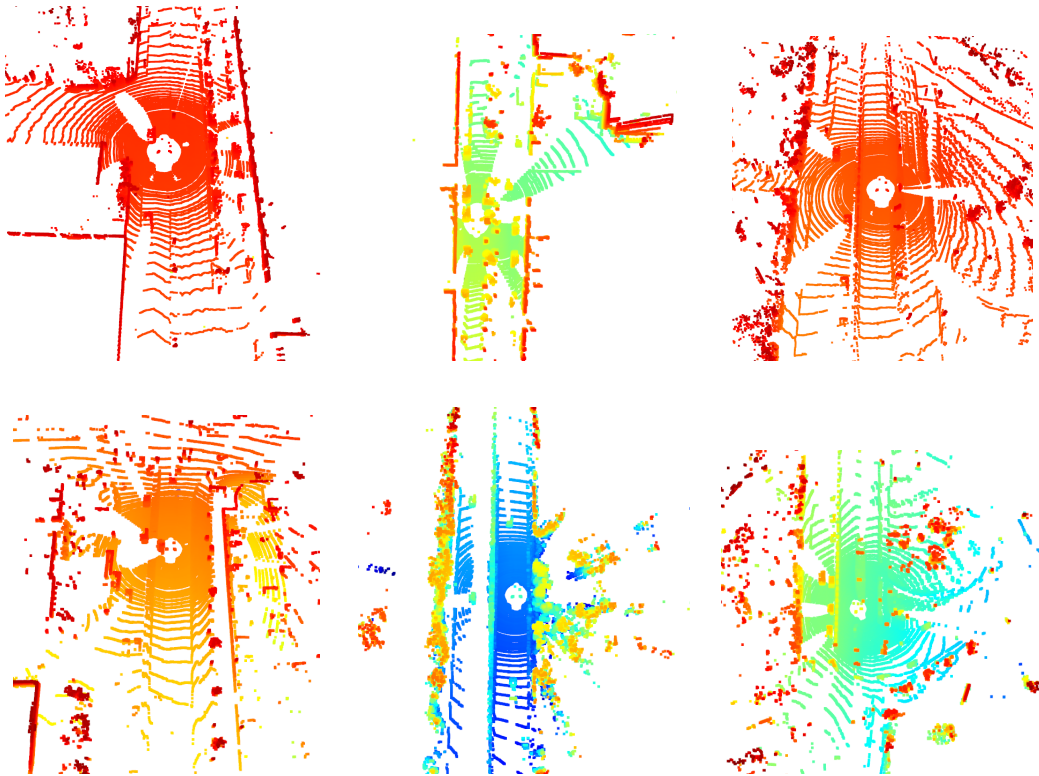


Tabla 4.1: Visualización de algunas nubes del USYD CAMPUS DATASET con Open3D.

4.3 Arquitectura del modelo

La arquitectura de nuestro proyecto es una variante del famoso PV-RCNN, un modelo que abstrae el conjunto de características del objeto a partir de los Voxels para realizar la detección de objetos en 3D, además de integrar una red de tipo PointNet.

4.3.1 PV-RCNN

Como se puede observar en la Figura 4.3, las nubes de puntos se voxelizan para que sirvan de entrada al encoder basado en convolución 3D para aprender las características semánticas y generar propuestas de objetos 3D. A continuación, los conjuntos de características aprendidos se agrupan en keypoints a través del módulo de abstracción. Por último, las características de los keypoints se agregan a la cuadrícula del módulo RoI para aprender características específicas para realizar un refinamiento de las predicciones y aumentar la fiabilidad.

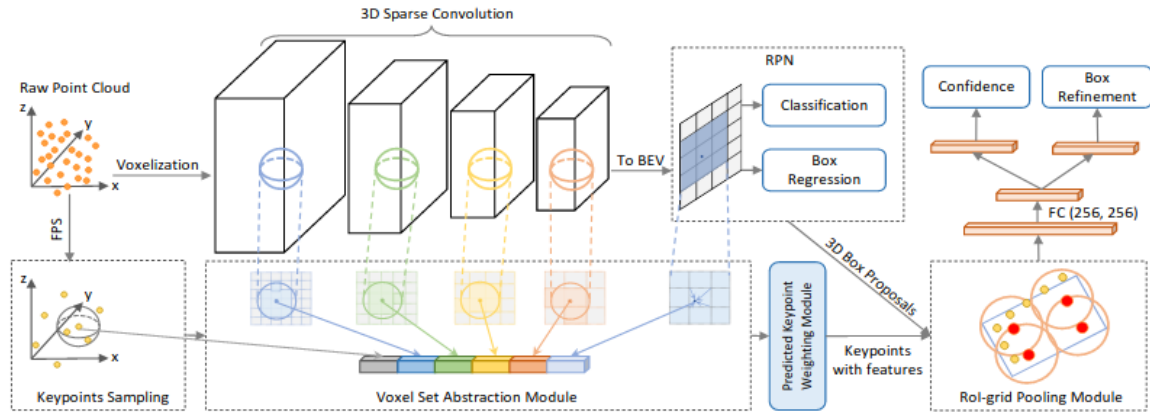


Figura 4.3: Arquitectura de PV-RCNN

Este modelo obtiene grandes resultados en la detección de coches y ciclistas, como puede observarse en la siguiente tabla:

Benchmark	Easy	Moderate	Hard
Car (Detection)	96.08 %	95.05 %	92.42 %
Car (Orientation)	96.07 %	94.90 %	92.22 %
Car (3D Detection)	90.14 %	81.88 %	77.15 %
Car (Bird's Eye View)	92.66 %	88.74 %	85.97 %
Cyclist (Detection)	85.76 %	79.22 %	72.35 %
Cyclist (Orientation)	85.58 %	78.44 %	71.60 %
Cyclist (3D Detection)	82.22 %	67.33 %	60.04 %
Cyclist (Bird's Eye View)	84.60 %	71.86 %	63.84 %

4.3.2 Nuestro modelo basado en PV-RCNN

Este modelo contiene dos nuevos módulos con respecto a la arquitectura del clásico PV-RCNN: el módulo de deformación adaptable y el módulo de fusión de contextos, los cuales pueden apreciarse en la Figura 4.4. Estos nuevos módulos permiten mejorar los resultados de detección 3D de objetos en nubes con densidades no uniformes y con objetos a largas distancias, y además puede abordar el desorden en las escenas de tráfico del mundo real.

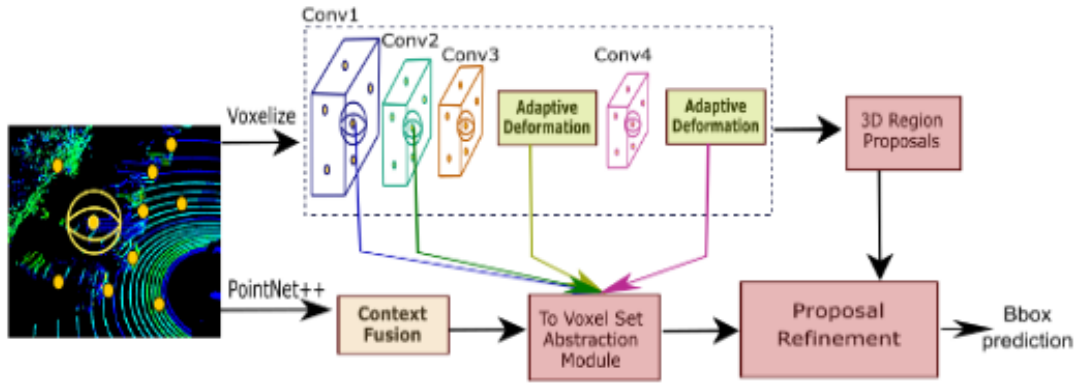


Figura 4.4: Arquitectura de nuestra aproximación

- **Módulo de deformación adaptable:** Este módulo se encarga de alinear los Keypoints hacia las características más ricas y discriminatorias, puesto que son las que más información nos pueden aportar.

Los n keypoints muestreados (mostrados en amarillo en la Figura 4.4) tienen una posición 3D

$$v_i$$

y un vector de características

$$f_i$$

correspondientes a las capas Conv3 y Conv4. A continuación, se calculan y actualizan las características

$$f'_i$$

de la siguiente manera:

$$f'_i = \frac{1}{n} \text{ReLU}(\sum_{j \in \mathcal{N}(i)} W_{offset}(f_i - f_j) \cdot (v_i - v_j))$$

donde

$$W_{offset}$$

es una matriz de pesos aprendidos.

Tras estos despejes, se obtienen las nuevas posiciones deformadas de los keypoints como:

$$v'_i = v_i + \tanh(W_{align}[f'_i])$$

donde

$$W_{align}$$

es también una matriz de pesos aprendidos.

Por último, se calculan las características para los keypoints deformados usando PointNet++, de manera similar al modelo PV-RCNN.

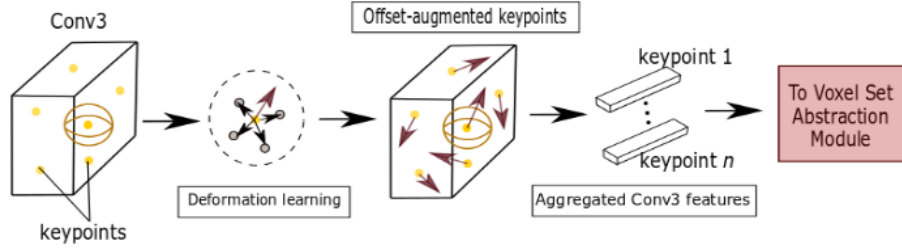


Figura 4.5: Arquitectura del módulo de deformación adaptable.

- **Módulo de fusión de contextos:** Este módulo selecciona dinámicamente las características más relevantes en función del contexto.

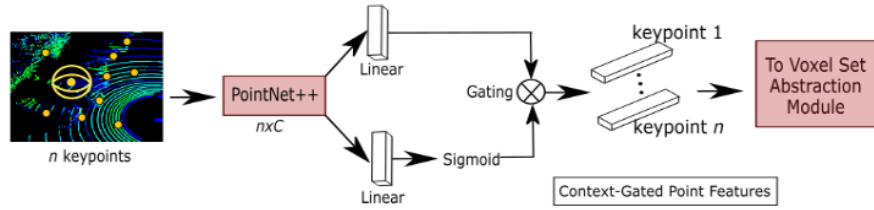


Figura 4.6: Arquitectura del módulo de fusión de contextos.

Usa el contexto para elegir las características más representativas y discriminatorias, resaltando las características de los objetos y eliminando el desorden.

Dada una característica del keypoint

$$f_i$$

la característica de modulación se obtiene como:

$$g = \sigma(W_{gate}f_i + b_{gate})$$

y la característica *Context-Gated* se calcula como:

$$f_i^g = g \odot W_{fc}f_i$$

4.4 ROS

ROS (Robot Operating System) es un meta-sistema operativo que provee de herramientas y librerías a los desarrolladores de software para crear aplicaciones robóticas. Proporciona los servicios básicos de un sistema operativo como pueden ser el control de dispositivos de bajo nivel, el paso de mensajes entre procesos, la abstracción del hardware y el manejo de paquetes. Actualmente, ROS solo está operativo sobre plataformas basadas en Unix.

ROS se va a utilizar en nuestro proyecto para el tratamiento y visualización de las nubes 3D del USYD CAMPUS DATASET, por lo que debemos entender los conceptos básicos y cómo funciona tanto el sistema de archivos como el grafo de computación que posee ROS.

4.4.1 Sistema de archivos

En el sistema de archivos de ROS encontramos los siguientes conceptos:

- **Paquetes:** Los paquetes son la unidad de organización principal de software que existe en ROS. Aquí podemos encontrar tanto datasets o archivos de configuración como procesos de ejecución propios de ROS.
- **Metapaquetes:** Son paquetes que representan un grupo de paquetes que están relacionados entre sí.
- **Manifiestos del paquete:** Incluyen metadatos sobre los paquetes de ROS, como por ejemplo su nombre, la descripción, las dependencias...
- **Tipos de mensajes:** Los mensajes son la información que un proceso envía a otro proceso y, por defecto, ROS tiene una gran cantidad de mensajes estandarizados.
- **Tipos de servicios:** Son las estructuras de petición y respuesta usadas por los servicios en ROS.

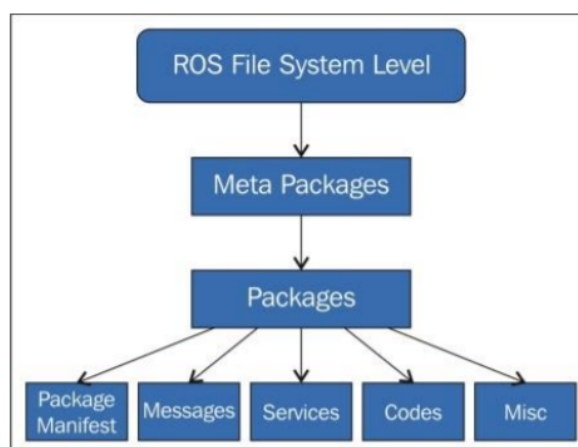


Figura 4.7: Sistema de archivos de ROS

4.4.2 Grafo de computación

El grafo de computación de ROS posee una arquitectura que consiste en una red descentralizada que permite la comunicación sin la necesidad de un intermediario. Los conceptos en los que se basa este grafo de computación son los siguientes:

- **Nodos:** Los nodos son los procesos donde se realiza el trabajo en ROS. Como ROS posee este tipo de grafo de computación, cualquier nodo puede comunicarse con otros nodos. Aunque un nodo puede controlar distintas funciones, se aconseja usar un nodo para cada funcionalidad, por ejemplo, un nodo controlaría únicamente el láser.
- **Master:** Este es un elemento esencial en ROS, ya que posibilita la comunicación entre nodos, servicios y mensajes.
- **Topics:** Este elemento proporciona la canalización de los mensajes en ROS. Es decir, un nodo puede publicar información en un *topic* y otro nodo se puede suscribir a ese mismo *topic* para leer los datos que se están enviando.
- **Mensajes:** Es la información que se envía de un nodo a otro.
- **Servicios:** Es una forma alternativa de comunicación síncrona en ROS que funciona a modo de cliente-servidor, es decir, un nodo realiza una petición a otro nodo y se bloquea hasta que se recibe la respuesta a la petición.
- **Bags:** Esta herramienta nos permite guardar mensajes para reproducirlos posteriormente. Esta es una función especialmente útil, ya que permite la simulación de la información que se grabó sin tener los elementos que la generan. Por ejemplo, el USYD CAMPUS DATASET ha sido grabado en este formato, lo que nos permite realizar simulaciones de todas las secuencias *a posteriori* sin necesidad de tener los sensores que se utilizaron.

A modo de ejemplo con nuestro caso, para poder visualizar las nubes obtenidas del USYD CAMPUS DATASET, tenemos que suscribirnos al *topic* por el que el nodo del sensor LiDAR está publicando los datos que obtiene.

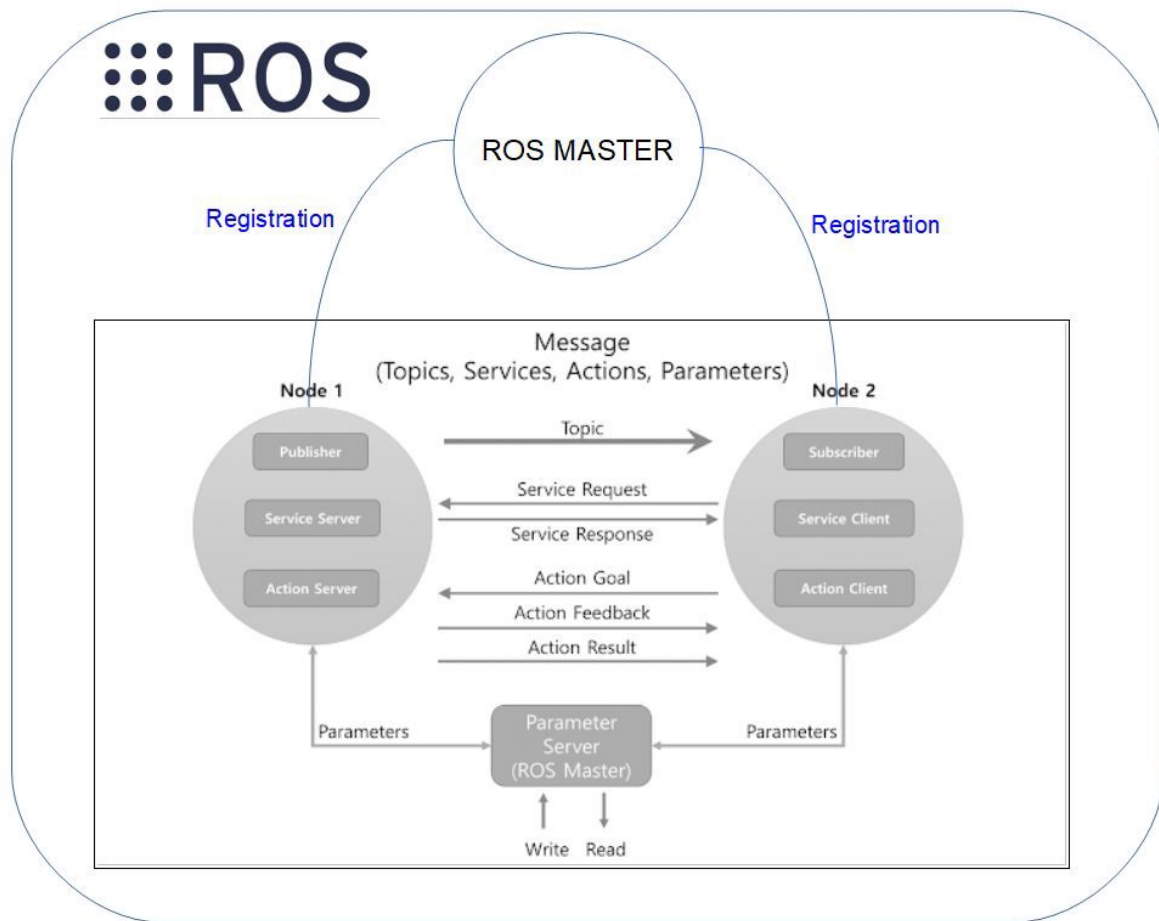


Figura 4.8: Grafo computacional de ROS

5 Desarrollo

En este apartado, voy a explicar cómo se ha llevado a cabo el desarrollo del proyecto y las herramientas necesarias para ello. El grueso del código está escrito en Python en un entorno conda, haciendo uso de diversos paquetes para las diferentes necesidades que se nos plantean, como puede ser PyTorch para la construcción y entrenamiento de la red y Open3D para la visualización de las nubes. Además, se realiza el preprocesamiento de las nubes del USYD CAMPUS DATASET para adaptarlas al formato que requiere nuestro modelo y su visualización mediante ROS y RViz.

Como se puede observar en la Figura 5.1, el desarrollo general consta de dos partes claramente diferenciadas. Por una parte, la implementación de la red, así como su entrenamiento y su test está desarrollado en Python mediante PyTorch, donde también se incluye el manejo de las nubes 3D del dataset de KITTI. Una vez que se obtiene el modelo con unos resultados buenos, es hora de probarlos en el dataset USYD CAMPUS DATASET, y observar su fiabilidad y robustez ante nuevos datos, con todo lo que ello supone (diferencia de densidad de las nubes, datos obtenidos con sensores distintos, cambios de escala, cambios del ángulo de visión...) e intentar realizar un etiquetado automático de los datos para futuros desarrollos en aprendizaje supervisado, puesto que este dataset no incluye actualmente el etiquetado de los objetos.

La vista detallada de cada una de estas partes del desarrollo las veremos a continuación.

5.1 Implementación de la red en PyTorch y prueba con el dataset KITTI

Para llevar a cabo el desarrollo de la arquitectura planteada anteriormente, debemos en primer lugar obtener la implementación de la red. Esta red neuronal está desarrollada usando la librería PyTorch, un paquete de Python diseñado para realizar cálculos numéricos haciendo uso de la programación de tensores, por lo que se utiliza habitualmente en el campo del machine learning y, concretamente, en el desarrollo de redes neuronales.

PyTorch dispone de una interfaz muy sencilla para la creación de redes neuronales, a pesar de trabajar directamente con tensores sin la necesidad de una librería a un nivel superior como pueda ser Keras o Tensorflow, y aunque es una librería muy reciente, dispone de una gran cantidad de manuales y tutoriales donde encontrar ejemplos, por lo que tiene una curva de aprendizaje fácilmente abordable para empezar a programar tus primeras redes. Dispone además de soporte para la ejecución en tarjetas gráficas, utilizando la API CUDA, la cual conecta la CPU con la GPU y ha sido desarrollada por NVIDIA.

Todos los datos referentes a la implementación y entrenamiento de la red, tales como número y tipos de capa, optimizador, learning rate... están definidos en un archivo de configuración, reduciendo enormemente el número de líneas de código que hay que escribir. Así pues, la construcción de la red quedaría tal y como podemos observar en el Código 5.1.

Código 5.1: Implementación del modelo

```

1  train_set, train_loader, train_sampler = build_dataloader(
2      dataset_cfg=cfg.DATA_CONFIG,
3      class_names=cfg.CLASS_NAMES,
4      batch_size=args.batch_size,
5      dist=dist_train, workers=args.workers,
6      logger=logger,
7      training=True,
8      merge_all_iters_to_one_epoch=args.merge_all_iters_to_one_epoch,
9      total_epochs=args.epochs
10 )
11
12 model = build_network(model_cfg=cfg.MODEL, num_class=len(cfg.CLASS_NAMES), dataset=
    ↪ train_set)

```

De esta sencilla manera ya tenemos construida nuestra red, a la que le faltaría añadir el optimizador (véase Código 5.2) y, tras esto, ejecutarla para que comience a entrenar (véase Código 5.3).

Código 5.2: Construcción del optimizador

```
optimizer = build_optimizer(model, cfg.OPTIMIZATION)
```

Código 5.3: Entrenamiento de la red

```

1  train_model(
2      model,
3      optimizer,
4      train_loader,
5      model_func=model_fn_decorator(),
6      lr_scheduler=lr_scheduler,
7      optim_cfg=cfg.OPTIMIZATION,
8      start_epoch=start_epoch,
9      total_epochs=args.epochs,
10     start_iter=it,
11     rank=cfg.LOCAL_RANK,
12     tb_log=tb_log,
13     ckpt_save_dir=ckpt_dir,
14     train_sampler=train_sampler,
15     lr_warmup_scheduler=lr_warmup_scheduler,
16     ckpt_save_interval=args.ckpt_save_interval,
17     max_ckpt_save_num=args.max_ckpt_save_num,
18     merge_all_iters_to_one_epoch=args.merge_all_iters_to_one_epoch
19 )

```

Una vez implementada nuestra red, deberemos modificar ciertos parámetros hasta obtener los mejores resultados posibles en entrenamiento y validación antes de probar el modelo con las nubes de testing. La variación de estos parámetros y cómo afectan a los resultados se expondrán más adelante en la sección de resultados.

Para poder observar los resultados sobre las nubes 3D, es decir, los bounding boxes sobre los objetos detectados, más allá de los resultados numéricos de porcentaje de acierto, se ha hecho

uso de ciertas funciones de visualización que incluye el proyecto OpenPCDet, el cual es un proyecto *Open Source* de detección de objetos 3D basados en LiDAR, las cuales me permiten visualizar los resultados pasándole el modelo y la nube sobre la que deseamos detectar los objetos.

5.2 Procesamiento del USYD CAMPUS DATASET y prueba del modelo

Anteriormente hemos comentado cómo ha sido obtenido el USYD CAMPUS DATASET. Sin embargo, tiene una peculiaridad que hace que debamos procesar los datos para poder usarlos. Las secuencias grabadas están en formato *.bag*, un tipo de archivo generado por un paquete nativo de ROS llamado *Rosbag* que permite realizar grabaciones de los datos que son publicados en los *topics* para poder reproducirlos posteriormente en ROS sin necesidad de tener conectado el sistema desde el que se obtuvieron los datos.

Así pues, tenemos un modelo que ha sido entrenado con las nubes de KITTI, las cuales están en formato *.bin*, pero unos nuevos datos que están en formato *.bag*. El objetivo está claro: obtener la nube de cada *frame* de las grabaciones del USYD CAMPUS DATASET y procesarlas para convertirlas a formato KITTI para que sea válido para nuestro modelo.

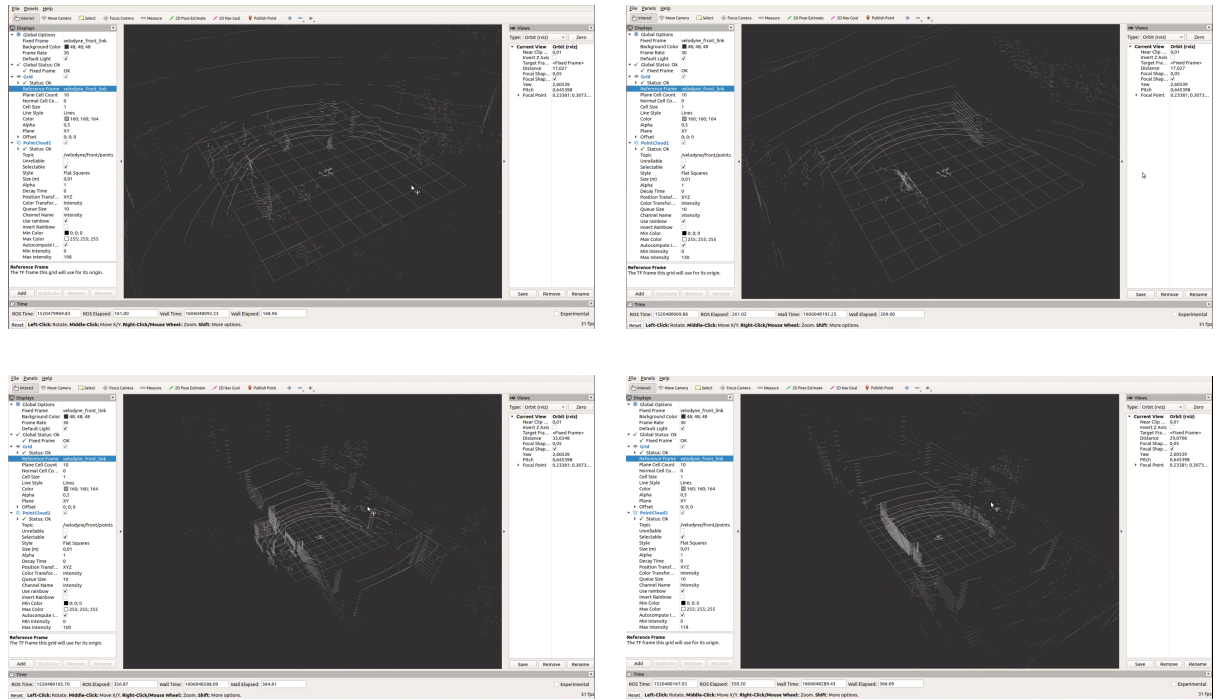


Tabla 5.1: Visualización del USYD CAMPUS DATASET en RViz.

Para ello, se ha hecho uso de dos paquetes de ROS: uno para pasar de formato *.bag* a *pcd* (nube de puntos), y otro para transformar esas nubes de puntos a formato binario (*.bin*).

5.2.1 Bag2Pcd

Este paquete nos permite decodificar los archivos grabados en ROS, dividiéndolos en nubes de puntos con formato *.pcd* y en imágenes 2D en formato *.png*. El funcionamiento básico de este paquete es el siguiente:

- Construimos el paquete mediante la instrucción *catkin_make* desde la carpeta *catkin_ws*
- Ejecutamos la instrucción *Roscore*
- Abrimos otra consola, y desde el *catkin_ws* ejecutamos la instrucción *./devel/lib/obstacle_detection/map_generate*
- Abrimos otra consola y ejecutamos la instrucción *roslaunch xxx bag -r 0.1*, donde *xxx.bag* es nuestra secuencia del USYD CAMPUS DATASET.
- Esto genera una nube de puntos por cada *frame* de la grabación y su correspondiente imagen.

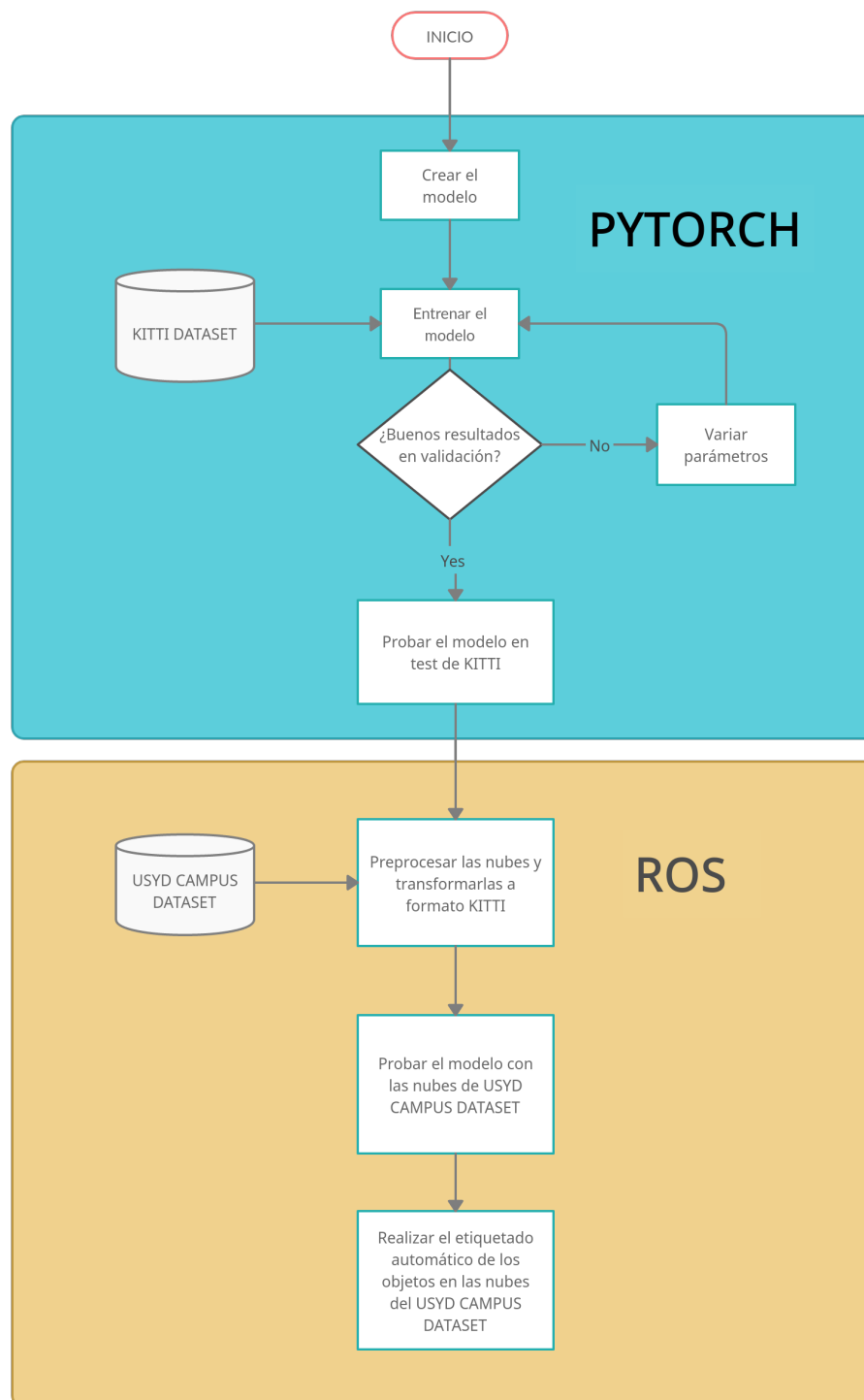
5.2.2 Pcd2Bin

Una vez tenemos las nubes de puntos en un formato amigable, debemos transformarlas a binario para facilitar la experimentación de detección de objetos 3D. Los archivos *.bin* generados contendrán 4 valores por cada punto de la nube 3D, los cuales corresponden con *xyz* y la intensidad. El funcionamiento básico de este paquete es el siguiente:

- Al igual que antes, construimos este paquete, en este caso mediante las instrucciones *cmake ..* y *make*, ejecutadas desde el archivo *CMakeFile*.
- Colocamos las nubes de puntos generadas anteriormente en la carpeta *pcd* y colocamos su ruta en el código fuente en *pcd2bin.cpp*
- Ejecutamos el programa mediante la instrucción *./pcd2bin*

Adicionalmente, se crean los ficheros *train.txt*, *trainval.txt* y *val.txt*, los cuales contienen una lista con los archivos *.bin* que habrán en cada una de las divisiones entrenamiento y validación.

En este punto ya tenemos los datos en un formato similar al del dataset de KITTI, lo que supone que podemos usarlos para probar nuestro modelo y ver los resultados que se obtienen, lo cual explicaré a continuación.

**Figura 5.1: Diagrama general del desarrollo**

6 Resultados

En este apartado voy a comentar los resultados obtenidos en este proyecto.

6.1 Explicación de los datos y los parámetros usados

En primer lugar, cabe mencionar cómo las etiquetas de los objetos están distribuidos estadísticamente a lo largo del dataset KITTI. Como podemos observar en la Tabla 6.1, hay dos clases claramente predominantes: coches y peatones. Además, podemos observar que estas apariciones se reparten entre objetos completos y objetos que no se observan completamente por diferentes motivos.

Por lo tanto, esto nos da una idea inicial de que el mayor porcentaje de acierto por parte del modelo va a ser en los objetos de las clases de coche y peatón, puesto que al aparecer un mayor número de veces en las nubes de entrenamiento, el sistema habrá sido capaz de generalizar mucho mejor con estos objetos que con otros que apenas aparecen en el dataset. Esto puede verse en la tabla 6.2, donde hay una mayor cantidad de coches por imagen (hay imágenes con hasta 10 coches) que de camiones por imagen (apenas hay camiones por imagen).

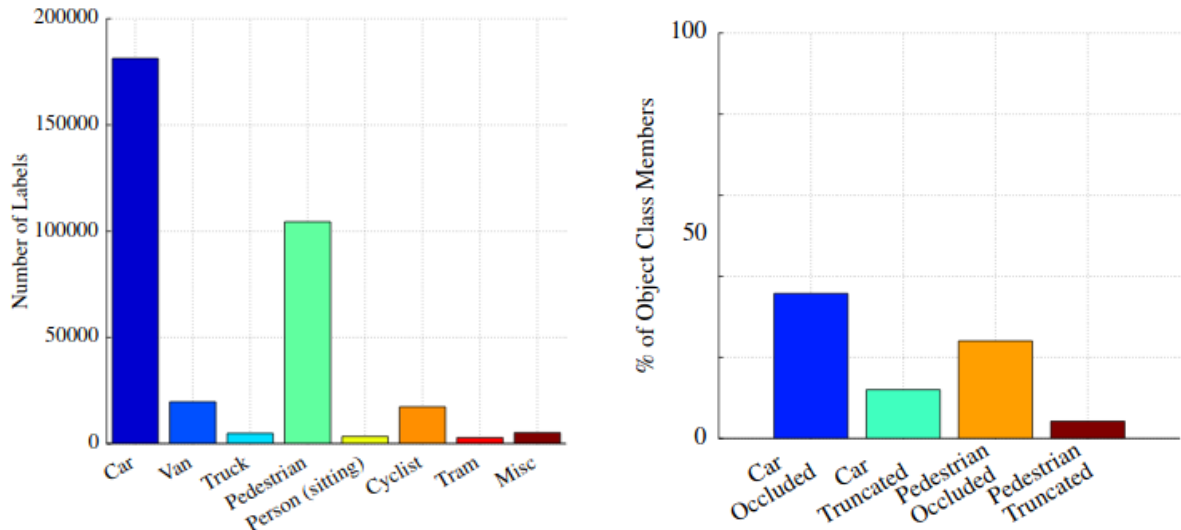


Tabla 6.1: Distribución de las etiquetas de los objetos en el Dataset KITTI.

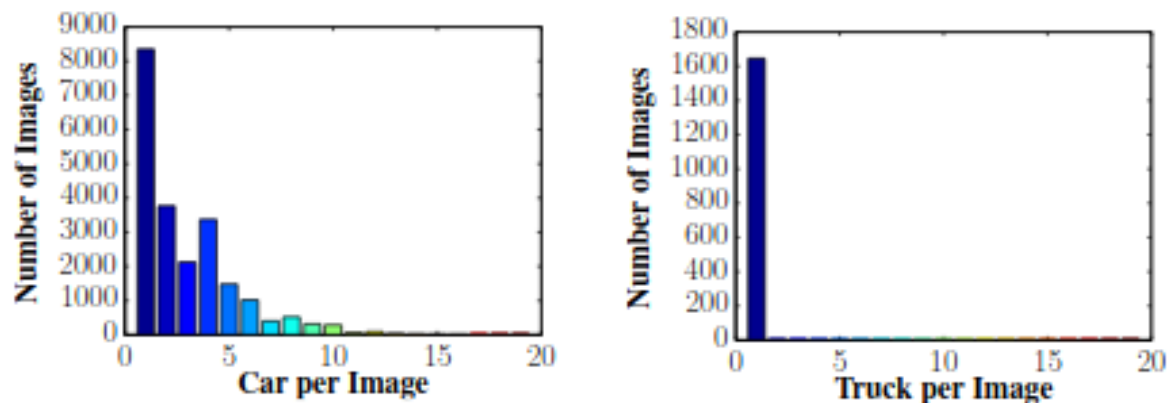


Tabla 6.2: Distribución de las apariciones de los objetos *Coche* y *Camión* en el Dataset KITTI.

Centrándonos ahora en la configuración de la red, los parámetros con los que se ha entrenado la red y mejores resultados se han obtenido han sido los siguientes:

- **Número de épocas** = 2000
- **Batch Size** = 32
- **Optimizador** = Adam
- **Learning Rate** = 0.01
- **Decay Step** = 200000

Voy a comentar brevemente el significado de cada uno de los hiper-parámetros mencionados para entender mejor cómo afecta cada uno de estos valores al resultado final.

- El número de épocas es la cantidad de iteraciones que la red va a estar entrenando. Esto significa que una cantidad de épocas bajo hará que la red no llegue a aprender, mientras que una cantidad de épocas excesivamente alto será una pérdida de tiempo y puede, en algunos casos, producir *Overfitting*¹.
- El Batch Size es la cantidad de ejemplos que la red predice en cada iteración, y se suele adaptar para que al final de cada época la red haya entrenado con todos los ejemplos.
- El optimizador es el descenso del gradiente, un algoritmo de aprendizaje que se aplica mientras la red está entrenando. En este caso, Adam es un tipo de este algoritmo.
- El Learning Rate nos muestra cómo se ajustan los pesos de la red al error. Esto significa que un Learning Rate muy bajo hace que la red aprenda muy lento, lo que puede provocar que el gradiente se quede estancado en mesetas o mínimos locales, y por lo

¹El *Overfitting* es el concepto que se produce cuando la red ha estado entrenando demasiado tiempo y, en vez de generalizar el aprendizaje, acaba aprendiendo los datos exactos del dataset, por lo que en el entrenamiento muestra resultados muy buenos pero al probarlo en el test produce resultados muy malos.

tanto no obtener los resultados óptimos. Por otra parte, un Learning Rate muy alto hará que el gradiente vaya avanzando con movimientos más grandes y provoque que no converja.

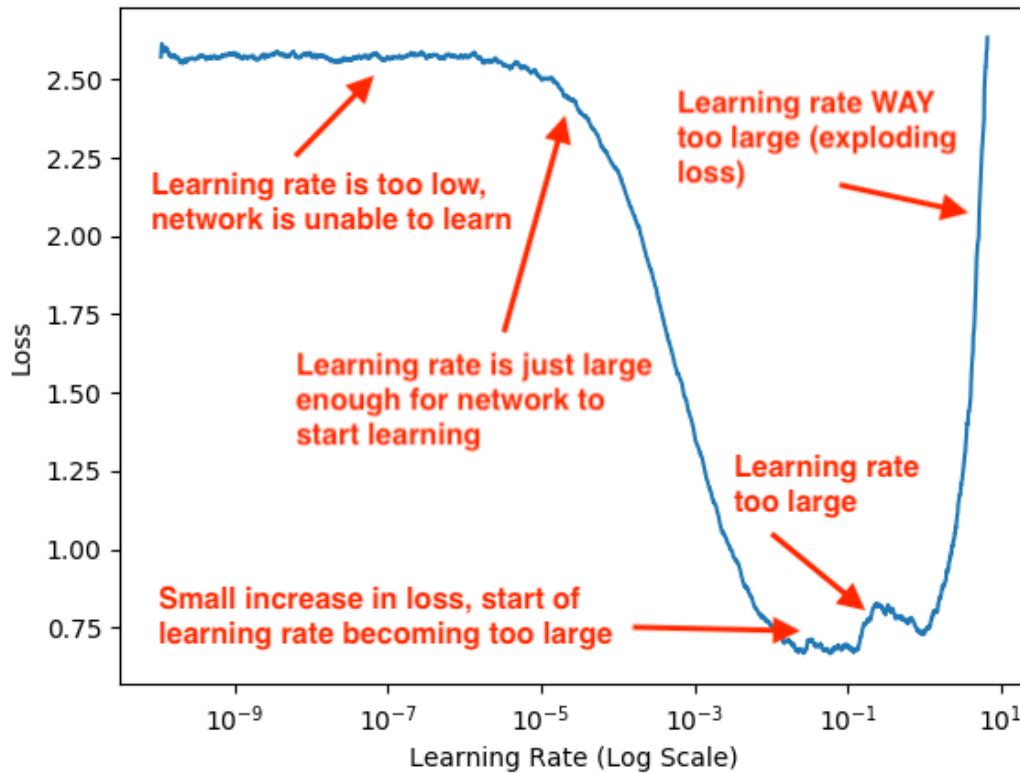


Figura 6.1: Impacto del *Learning Rate* en el aprendizaje

6.2 Visualización de los resultados

En este apartado voy a mostrar los resultados obtenidos sobre las nubes 3D, es decir, los *Bounding Boxes* obtenidos por el modelo sobre los objetos detectados en cada una de las escenas. Cabe mencionar que los objetos de la clase *Coche* se representan con un *Bounding Box* de color verde, los objetos de la clase *Peatón* con un *Bounding Box* azul, y los objetos de la clase *Ciclista* con un *Bounding Box* amarillo. Muestro estas clases porque son las principales del dataset, y por lo tanto aparecen en la mayoría de escenas.

Además, para facilitar la lectura de los resultados, muestro la nube 3D con los resultados junto con su correspondiente imagen 2D para que se pueda ver de un simple vistazo si la detección de objetos ha sido correcta o no.

6.2.1 Resultados en el dataset KITTI

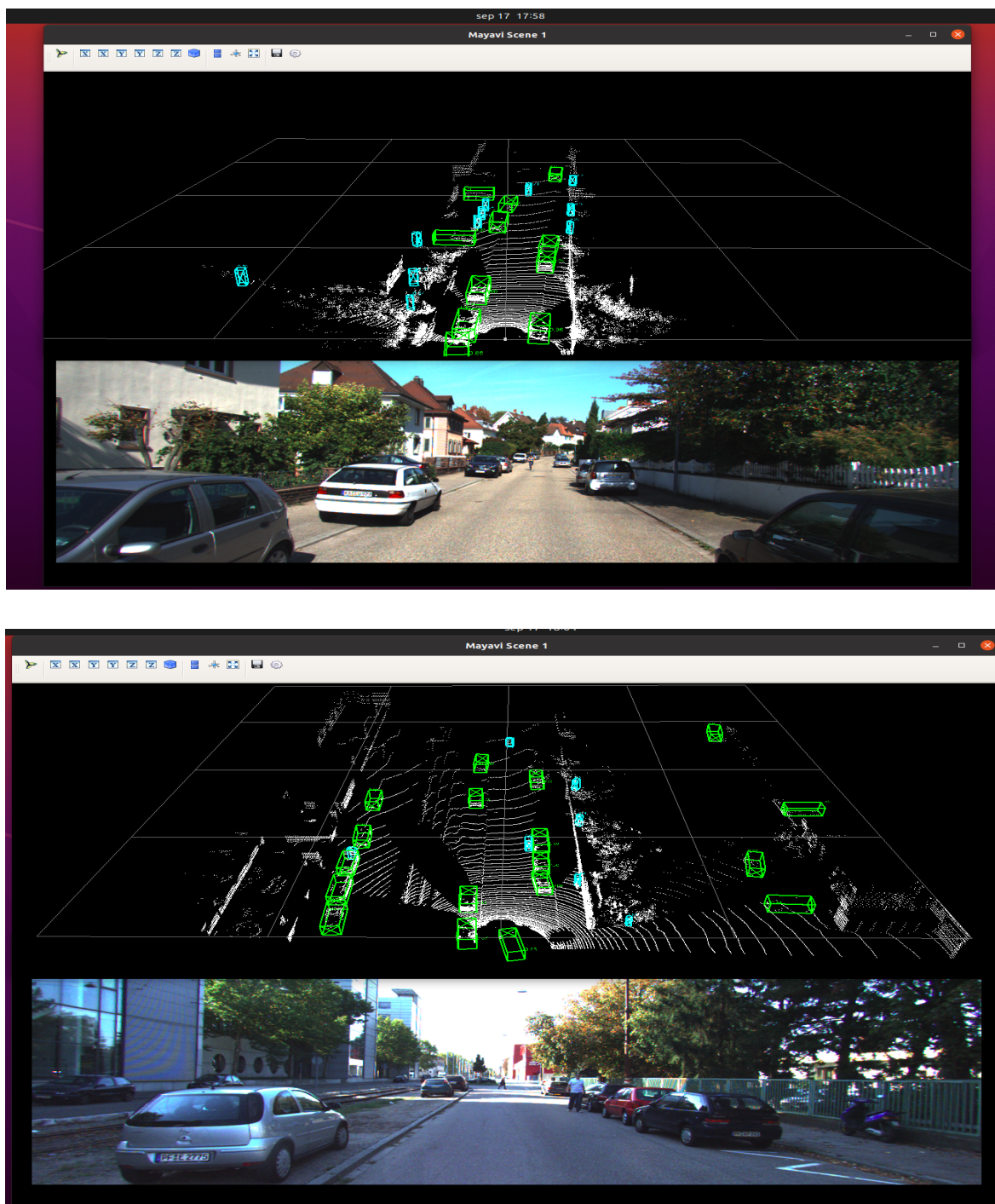


Tabla 6.3: Resultados de la detección en escenas con coches y peatones

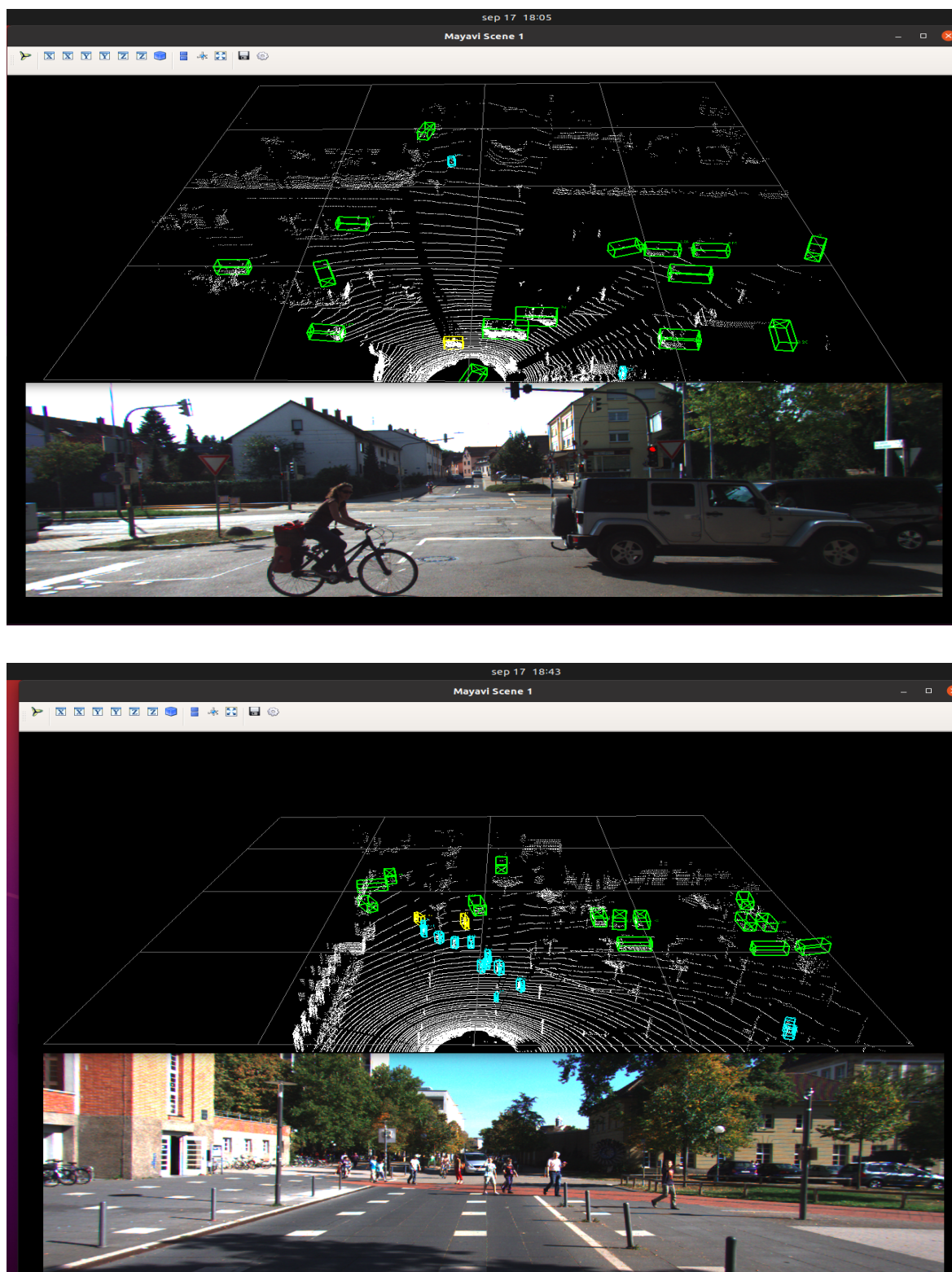


Tabla 6.4: Resultados de la detección en escenas con ciclistas

Como podemos observar en las Tablas 6.3 y 6.4, los resultados obtenidos son bastante buenos, puesto que detecta correctamente las principales clases del dataset y, en la mayoría de casos, a una distancia muy lejana. Los resultados del modelo por clase están recogidos en la siguiente tabla:

Coche	Peatón	Ciclista
83.33 %	58.84 %	73.91 %

Tabla 6.5: Resultados de detección de objetos del modelo de las principales clases del dataset KITTI

6.2.2 Resultados en USYD CAMPUS DATASET

Una vez validado el modelo con el dataset KITTI, procedemos a probarlo con el nuevo dataset.

6.2.3 Problemas del modelo con el USYD CAMPUS DATASET

A pesar de haber preprocesado las nubes 3D para adaptarlas al formato que necesita nuestro modelo, las primeras pruebas resultan fallidas. El modelo no detecta prácticamente ningún objeto en las escenas, como puede observarse en el ejemplo tomado en la Figura 6.2

Tras esta primera toma de contacto, procedemos a investigar las posibles causas por las que el modelo no funciona correctamente con este dataset. Tras barajar varias opciones, observamos que el sensor con el que se han obtenido las nubes 3D es distinto que el que se ha usado en el de KITTI, lo que afecta tanto a la densidad de puntos de la nube como a la escala en la que se encuentran los objetos. Este sensor es el Velodyne VLP-16, un pequeño LiDAR 3D que tiene mediciones en tiempo real, 360° y reflectividad calibrada. Sin embargo, posee una densidad de puntos 3D cuatro veces más pequeña que el sensor utilizado en KITTI y los datos están tomados en una escala diferente. Esto afectaría a cómo el modelo detecta los objetos, y por ello tiene resultados tan malos de primeras. Para confirmar esta teoría, reducimos mediante voxelizado las nubes 3D del dataset KITTI de la parte de test a una cuarta parte (para asemejarlas a las del USYD CAMPUS DATASET) y se las pasamos al modelo, para comprobar si con estas nubes con las que antes funcionaba correctamente, ahora también funciona bien o, por el contrario, tampoco funciona. Así pues, las nubes pasan de tener unos 100-120 mil puntos a tener unos 25-30 mil puntos.

Al probar el modelo con estas nubes de KITTI reducidas, observamos que, efectivamente, se obtienen resultados muy pobres, similares a los obtenidos con el USYD CAMPUS DATASET.

6.2.4 Soluciones al problema presentado

Tras definir el problema, intentamos encontrar la solución para que el modelo funcione correctamente. Como sabemos que la densidad está afectando a los resultados del modelo, decidimos reentrenar el modelo, pero en este caso con las nubes KITTI reducidas que hemos obtenido anteriormente, para posteriormente volver a pasar el USYD CAMPUS DATASET al modelo y observar si se producen mejoras, teniendo así dos versiones del modelo: uno que funcione con nubes que tienen aproximadamente 100 mil puntos, y otro que funcione con una menor densidad de aproximadamente 25 mil puntos.

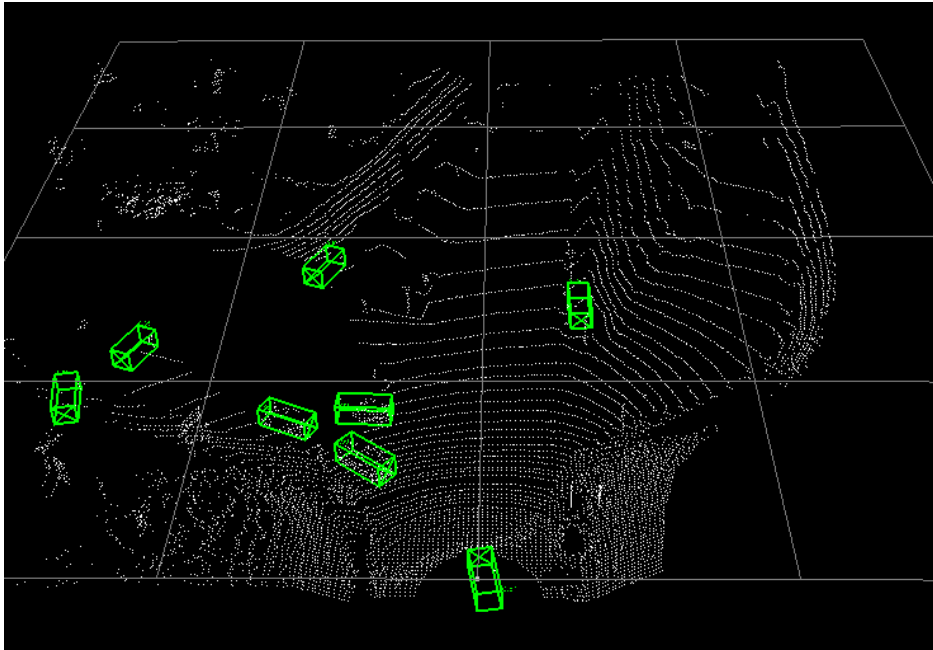


Figura 6.3: Resultados del modelo en USYD CAMPUS DATASET tras reentrenar el modelo. Ejemplo 1.

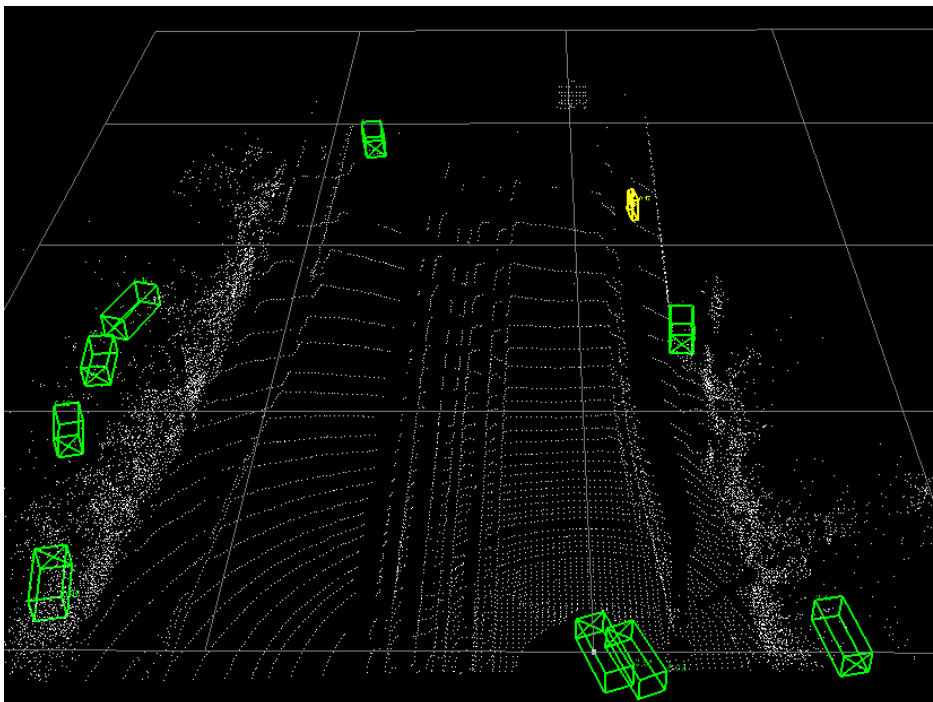


Figura 6.4: Resultados del modelo en USYD CAMPUS DATASET tras reentrenar el modelo. Ejemplo 2.

7 Conclusiones

En primer lugar, quiero recordar la idea principal en la que se centra este proyecto: la detección de objetos 3D mediante LiDAR y Deep Learning para ayudar a los sistemas de conducción autónoma.

Para alcanzar este objetivo se propusieron diferentes ideas, y se estudió una serie de proyectos para elegir uno sobre el que basarnos y empezar nuestro proyecto. Algunos de estos proyectos fueron *SalsaNext* o *Avod*, aunque fueron descartados por distintos motivos para, finalmente, elegir el proyecto *PV-RCNN*.

Una vez tenemos el punto de partida, comienza el desarrollo del proyecto, el cual podríamos dividir en dos: una primera parte con objetivos claramente marcados y un carácter puramente de desarrollo, y una segunda parte con un carácter de investigación con objetivos variables en función de los problemas y resultados obtenidos.

En la primera parte de este proyecto, nos hemos dedicado a implementar y probar el modelo definido en la arquitectura utilizando las nubes 3D del dataset KITTI, siendo el objetivo obtener un modelo de detección de objetos 3D con un porcentaje de acierto alto. El desarrollo de esta parte se ha producido de una manera fluida y con unos resultados muy buenos y relativamente rápidos, incluso rozando el 85% de acierto en los objetos de la clase *Coche*, debido en parte a que se han utilizado técnicas, herramientas y datos que tienen un alto soporte por parte de la comunidad científica, lo que hace más rápido el aprendizaje por mi parte de las técnicas usadas y el manejo de los datos.

Para la segunda parte se nos presentaba un amplio abanico de opciones por las que continuar el desarrollo del proyecto. Tras barajar estas opciones, se decide usar el USYD CAMPUS DATASET e investigar posibles aplicaciones del modelo sobre este dataset. Por tanto, se propone como primer objetivo la visualización de este dataset, lo cual se cumple utilizando ROS, puesto que es el formato que tienen los datos del dataset en crudo.

El siguiente objetivo era transformar estos datos a formato KITTI para poder usarlo con nuestro modelo, lo cual realizamos, como hemos explicado anteriormente, con una serie de paquetes de ROS. Tras esto, debíamos solucionar los problemas que se nos plantearan a la hora de probar el modelo con este dataset, como por ejemplo la diferencia de densidades en las nubes, lo cual solventamos reentrenando la red con nubes de KITTI reducidas una cuarta parte mediante Open3D.

El resultado con este dataset finalmente es bueno, aunque obteniendo un porcentaje de acierto entre un 10% y un 15% menor que la primera versión del modelo con el dataset de KITTI, algo asumible y entendible debido a la baja cantidad de puntos que tienen estas nubes. Además, el tiempo de ejecución de esta versión del modelo es similar a la del modelo original, ya que prácticamente no han habido modificaciones de los parámetros.

Por último, se plantea el objetivo de usar el modelo, el cual ya obtiene buenos resultados de detección de los objetos 3D en el USYD CAMPUS DATASET, para realizar el etiquetado automático de los objetos presentes en estas nubes de puntos, con el fin de evitar el etiquetado

manual de los miles de objetos que hay en las 52 secuencias de este dataset, con todo lo que ello supone en cuanto a recursos de tiempo y personal.

Bibliografía

- Alexnet: The architecture that challenged cnns.* (2019, January). Descargado 3/01/2021, de <https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951>
- Autonomous driving.* (2016, January). Descargado 3/01/2021, de https://www.sae.org/standards/content/j3016_201609/
- Autonomous driving levels.* (2016, January). Descargado 3/01/2021, de <https://www.km77.com/reportajes/varios/conduccion-autonoma-niveles>
- Bhattacharyya, P., y Czarnecki, K. (2020). Deformable pv-rcnn: Improving 3d object detection with learned deformations. *arXiv:1506.02640*.
- Engelcke, M., Rao, D., Wang, D. Z., Tong, C. H., y Posner, I. (2016). Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. *arXiv:1609.06666*.
- Geiger, A., Lenz, P., Stiller, C., y Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.
- Graham, B., Engelcke, M., y van der Maaten, L. (2018). 3d semantic segmentation with submanifold sparse convolutional networks. *CVPR*.
- Huang, Y., y Chen, Y. (2020). Autonomous driving with deep learning: A survey of state-of-art technologies. *arXiv:2006.06091*.
- Krizhevsky, A. (2012). Imagenet classification with deep convolutional neural networks.
- Pang, S., Morris, D., y Radha, H. (2020). Clocs: Camera-lidar object candidates fusion for 3d object detection. En *2020 ieee/rsj international conference on intelligent robots and systems (iros)*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. En H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, y R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Descargado de <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- Qi, C. R., Liu, W., Wu, C., Su, H., y Guibas, L. J. (2017). Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*.

-
- Ramachandran, P., y Varoquaux, G. (2011). Mayavi: 3d visualization of scientific data. En *Ieee computing in science & engineering*.
- Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2015). You only look once: Unified, real-time object detection. *arXiv:1506.02640*.
- Ros documentation*. (2019, January). Descargado 3/01/2021, de <http://wiki.ros.org/Documentation>
- Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., y Li, H. (2020). Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. En *Cvpr*.
- Tesla won't use lidar*. (2020, January). Descargado 3/01/2021, de <https://towardsdatascience.com/why-tesla-wont-use-lidar-57c325ae2ed5>
- Top companies in the global automotive lidar sensors market*. (2020, January). Descargado 3/01/2021, de <https://leddartech.com/top-companies-global-automotive-lidar-sensors-market/>
- Understanding the ros file system level*. (2019, January). Descargado 3/01/2021, de https://subscription.packtpub.com/book/hardware_and_creative/9781783551798/1/ch01lv11sec11/understanding-the-ros-file-system-level
- Zhang, Q., Jiang, Z., Lu, Q., Han, J., Zeng, Z., Gao, S.-H., y Men, A. (2020). Split to be slim: An overlooked redundancy in vanilla convolution. En *International joint conference on artificial intelligence (ijcai)*.
- Zhou, Q.-Y., Park, J., y Koltun, V. (2018). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.
- Zhou, W., Perez, J. S. B., Alvis, C. D., Shan, M., Worrall, S., Ward, J., y Nebot, E. (2019). The usyd campus dataset.. Descargado de <https://dx.doi.org/10.21227/sk74-7419> doi: 10.21227/sk74-7419
-